## VISION & MISSION OF THE INSTITUTE

### VISION

To induce higher planes of learning by imparting technical education with

- International standards
- Applied research
- Creative Ability
- Value based instruction and to emerge as a premiere institute.

### MISSION

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- Innovative Research and development
- Industry Institute Interaction
- Empowered Manpower

## VISION & MISSION OF THE DEPARTMENT

### VISION

To be a recognized Computer Science and Engineering hub striving to meet the growing needs of the Industry and Society.

### MISSION

M1: Imparting Quality Education through state-of-the-art infrastructure with industry Collaboration

M2: Enhance Teaching Learning Process to disseminate knowledge.

M3: Organize Skill based, Industrial and Societal Events for overall Development.

## EXERCISE-1(a)

**Course Name**: Angular JS

**Module Name:** Angular Application Setup Observe the link http://localhost:4200/welcome on which the mCart application is running. Perform the below activities to understand the features of the application.

### INSTALLING ANGULAR.JS IN VISUALSTUDIO

| **REQUIREMENTS** | **DETAILS** |
|---|---|
| Node.js | Angular requires an active LTS or maintenance LTS version of Node.js.<br><br>For information see the version compatibility guide.<br><br>For more information on installing Node.js, see nodejs.org. If you are unsure what version of Node.js runs on your system, run node -v in a terminal window. |
| npm package manager | Angular, the Angular CLI, and Angular applications depend on npm packages for many features and functions. To download and install npm packages, you need an npm package manager. This guide uses the npm client command line interface, which is installed with Node.js by default. To check that you have the npm client installed, run npm -v in a terminal window. |

Install the Angular CLI

You can use the Angular CLI to create projects, generate application and library code, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

To install the Angular CLI, open a terminal window and run the following command:

**npm install -g @angular/cli**

On Windows client computers, the execution of PowerShell scripts is disabled by default. To allow the execution of PowerShell scripts, which is needed for npm global binaries, you must set the following execution policy:

**Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned**

Carefully read the message displayed after executing the command and follow the instructions. Make sure you understand the implications of setting an execution policy.

Create a workspace and initial application

You develop apps in the context of an Angular workspace.

To create a new workspace and initial starter app:

1. Run the CLI command ng new and provide the name my-app, as shown here:

   content_copyng new my-app

2. The ng new command prompts you for information about features to include in the initial app. Accept the defaults by pressing the Enter or Return key.

The Angular CLI installs the necessary Angular npm packages and other dependencies. This can take a few minutes.

The CLI creates a new workspace and a simple Welcome app, readyto run.

Run the application

The Angular CLI includes a server, for you to build and serve your app locally.

1. Navigate to the workspace folder, such as my-app.
2. Run the following command:

   **my-app**

   **ng serve --open**

The ng serve command launches the server, watches your files, and rebuilds the app as you make changes to those files.

The --open (or just -o) option automatically opens your browser to http://localhost:4200/.

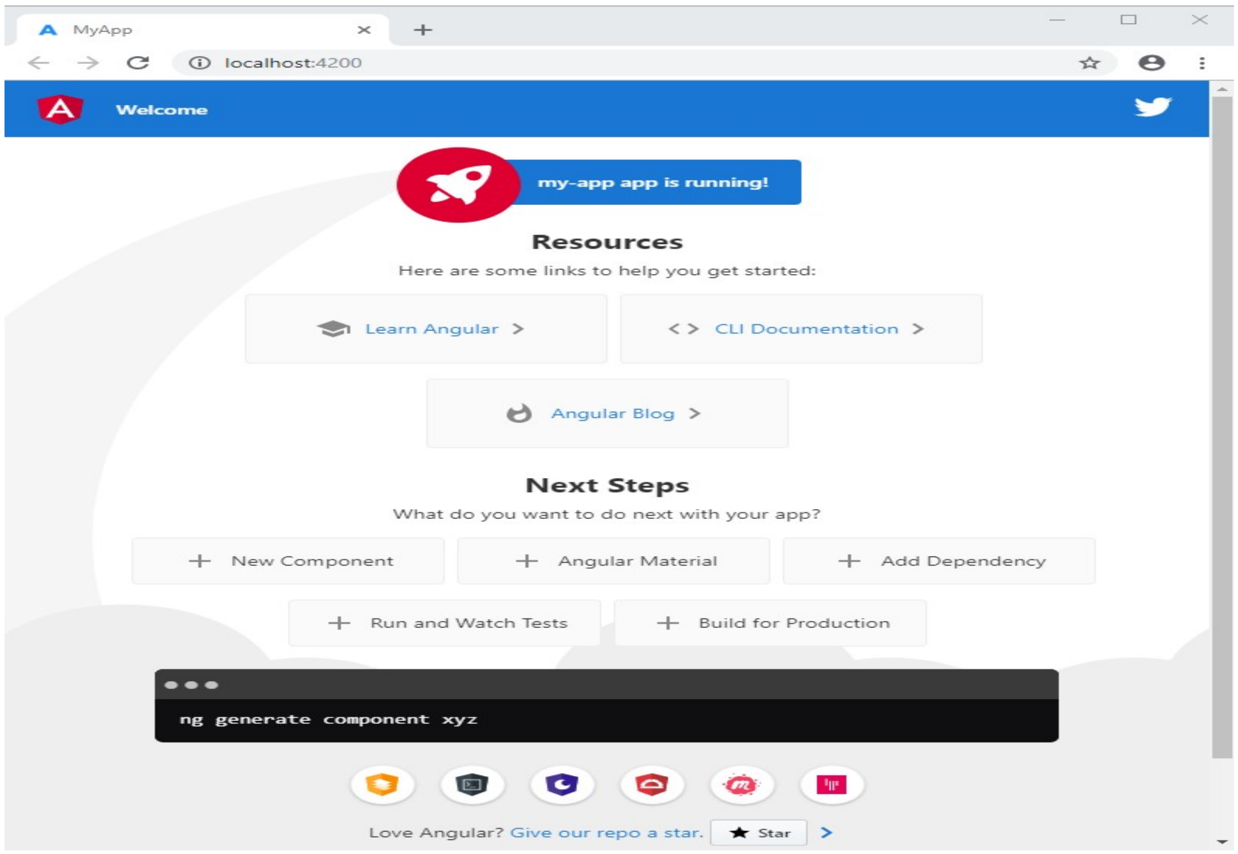
**Output:**

PS C:\Users\SRK\Desktop\cseb rockers> **npm install -g @angular/cli**

PS C:\Users\SRK\Desktop\cseb rockers> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

PS C:\Users\SRK\Desktop\cseb rockers> ng new my-app

```
Node.js version v19.8.1 detected.

Odd numbered Node.js versions will not enter LTS status and should not be used for
production. For more information, please see https://nodejs.org/en/about/releases/.

? Would you like to add Angular routing? Yes

? Which stylesheet format would you like to use? Less   [ http://lesscss.org

                  ]

CREATE my-app/angular.json (2874 bytes)

CREATE my-app/package.json (1037 bytes)

CREATE my-app/README.md (1059 bytes)

CREATE my-app/tsconfig.json (901 bytes)

CREATE my-app/.editorconfig (274 bytes)

CREATE my-app/.gitignore (548 bytes)

CREATE my-app/tsconfig.app.json (263 bytes)

CREATE my-app/tsconfig.spec.json (273 bytes)

CREATE my-app/.vscode/extensions.json (130 bytes)

CREATE my-app/.vscode/launch.json (470 bytes)

CREATE my-app/.vscode/tasks.json (938 bytes)

CREATE my-app/src/favicon.ico (948 bytes)

CREATE my-app/src/index.html (291 bytes)

CREATE my-app/src/styles.less (80 bytes)

CREATE my-app/src/app/app-routing.module.ts (245 bytes)

CREATE my-app/src/app/app.module.ts (393 bytes)

CREATE my-app/src/app/app.component.html (23115 bytes)

CREATE my-app/src/app/app.component.spec.ts (991 bytes)

CREATE my-app/src/app/app.component.ts (211 bytes)
CREATE my-app/src/app/app.component.less (0 bytes)

CREATE my-app/src/assets/.gitkeep (0 bytes)

✔ Packages installed successfully.

    Directory is already under version control. Skipping initialization of git.

PS C:\Users\SRK\Desktop\cseb rockers> cd my-app

>> ng serve --open

Node.js version v19.8.1 detected.

Odd numbered Node.js versions will not enter LTS status and should not be used for
production. For more information, please see https://nodejs.org/en/about/releases/.

? Would you like to share pseudonymous usage data about this project with the Angular
Team

at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more

details and how to change this setting, see https://angular.io/analytics. Yes

Thank you for sharing pseudonymous usage data. Should you change your mind, the
following

command will disable this feature entirely:

    ng analytics disable

Global setting: enabled

Local setting: enabled

Effective status: enabled

✔ Browser application bundle generation complete.

Initial Chunk Files   | Names         | Raw Size

vendor.js             | vendor        |  2.28 MB |

polyfills.js          | polyfills     | 333.14 kB |

styles.css, styles.js | styles        | 230.67 kB |

main.js               | main          |  48.08 kB |
runtime.js        | runtime      |  6.51 kB |

              | Initial Total |  2.89 MB

Build at: 2023-07-24T04:42:02.409Z - Hash: 8542936140e45a51 - Time: 7980ms

** Angular Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200/ **

√ Compiled successfully.
```



```
ng generate component xyz
```

**Exercise-1(b)**

**Course Name: Angular JS**

**Module Name:** Components and Modules Create a new component called hello and render Hello Angular on the page.

**Process :**

**Step 1** - Test the default app

In this step, after you download the default starting app, you build the default Angular app. This confirms that your development environment has what you need to continue the tutorial.

In the Terminal pane of your IDE:

1. In your project directory, navigate to the first-app directory.
2. Run this command to install the dependencies needed to run the app.

   ```
   npm install
   ```

3. Run this command to build and serve the default app.

   ```
   ng serve
   ```

   The app should build without errors.

4. In a web browser on your development computer, open http://localhost:4200.
5. Confirm that the default web site appears in the browser.
6. You can leave ng serve running as you complete the next steps.

**Step 2** - Review the files in the project

In this step, you get to know the files that make up a default Angular app.

In the Explorer pane of your IDE:

1. In your project directory, navigate to the first-app directory.
2. Open the src directory to see these files.
   a. In the file explorer, find the Angular app files (/src).
      i. index.html is the app's top level HTML template.
      ii. style.css is the app's top level style sheet.
      iii. main.ts is where the app start running.
      iv. favicon.ico is the app's icon, just as you would find in any web site.
   b. In the file explorer, find the Angular app's component files (/app).

i. app.component.ts is the source file that describes the app-root component. This is the top-level Angular component in the app. A component is the basic building block of an Angular application. The component description includes the component's code, HTML template, and styles, which can be described in this file, or in separate files.

   In this app, the styles are in a separate file while the component's code and HTML template are in this file.

ii. app.component.css is the style sheet for this component.
iii. New components are added to this directory.

c. In the file explorer, find the image directory (/assets) that contains images used by the app.
d. In the file explorer, find the files and directories that an Angular app needs to build and run, but they are not files that you normally interact with.
   i. .angular has files required to build the Angular app.
   ii. .e2e has files used to test the app.
   iii. .node_modules has the node.js packages that the app uses.
   iv. angular.json describes the Angular app to the app building tools.
   v. package.json is used by npm (the node package manager) to run the finished app.
   vi. tsconfig.* are the files that describe the app's configuration to the TypeScript compiler.

After you have reviewed the files that make up an Angular app project, continue to the next step.

**Step 3** - Create Hello World

In this step, you update the Angular project files to change the displayed content.

In your IDE:

1. Open first-app/src/index.html.
2. In index.html, replace the <title> element with this code to update the title of the app.

   Replace in src/index.html

   **<title>Homes</title>**

   Then, save the changes you just made to index.html.

3. Next, open first-app/src/app/app.component.ts.
4. In app.component.ts, in the @Component definition, replace the template line with this code to change the text in the app component.

   Replace in src/app/app.component.ts

   **template: `<h1>Hello world!</h1>`,**

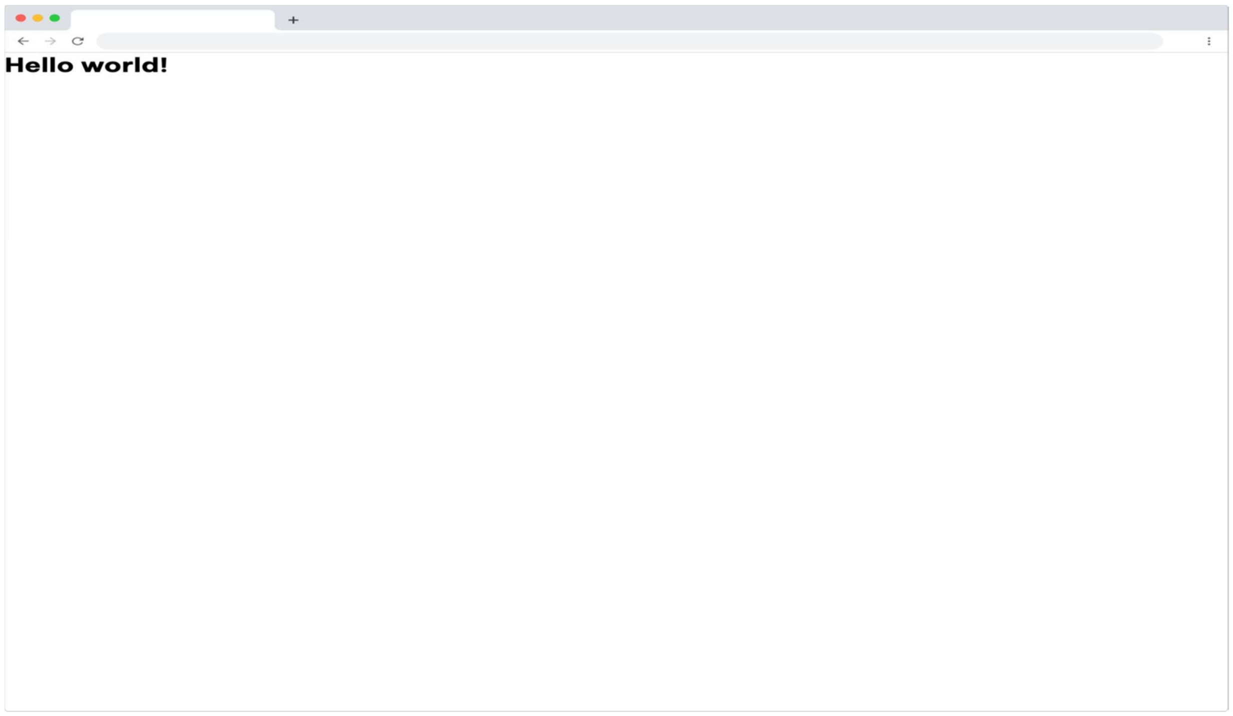
5. In app.component.ts, in the AppComponent class definition, replace the title line with this code to change the component title.

   Replace in src/app/app.component.ts

   **title = 'homes';**

   Then, save the changes you made to app.component.ts.

6. If you stopped the ng serve command fromstep 1, in the Terminal window of your IDE, run ng serve again.
7. Open your browser and navigate to localhost:4200 and confirm that the app builds without error and displays *Hello world* in the title and body of your app:



Hello world!

**Program:**

**Helloworld.js**

**Index.html:**

```
<!doctype html>

<html lang="en">

<head>
```

```
  <meta charset="utf-8">

  <title>Default</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="icon" type="image/x-icon" href="favicon.ico">

  <link rel="preconnect" href="https://fonts.googleapis.com">

  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

  <link href="https://fonts.googleapis.com/css2?family=Be+Vietnam+Pro:ital,wght@0,40
0;0,700;1,400;1,700&display=swap" rel="stylesheet">

</head>

<body>

  <app-root></app-root>

</body>

</html>
```

**Main.ts:**

```
import { bootstrapApplication,provideProtractorTestingSupport } from '@angular/platfor
m-browser';

import { AppComponent } from './app/app.component';



bootstrapApplication(AppComponent,

    {providers: [provideProtractorTestingSupport()]})

  .catch(err => console.error(err));
```

**Style.css:**

```
 {
  margin: 0;
  padding: 0;
}

body {
  font-family: 'Be Vietnam Pro', sans-serif;
}
:root {
  --primary-color: #605DC8;
  --secondary-color: #8B89E6;
  --accent-color: #e8e7fa;
  --shadow-color: #E8E8E8;
}

button.primary {
  padding: 10px;
  border: solid 1px var(--primary-color);
  background: var(--primary-color);
  color: white;
  border-radius: 8px;
}
```

**App.component.css**

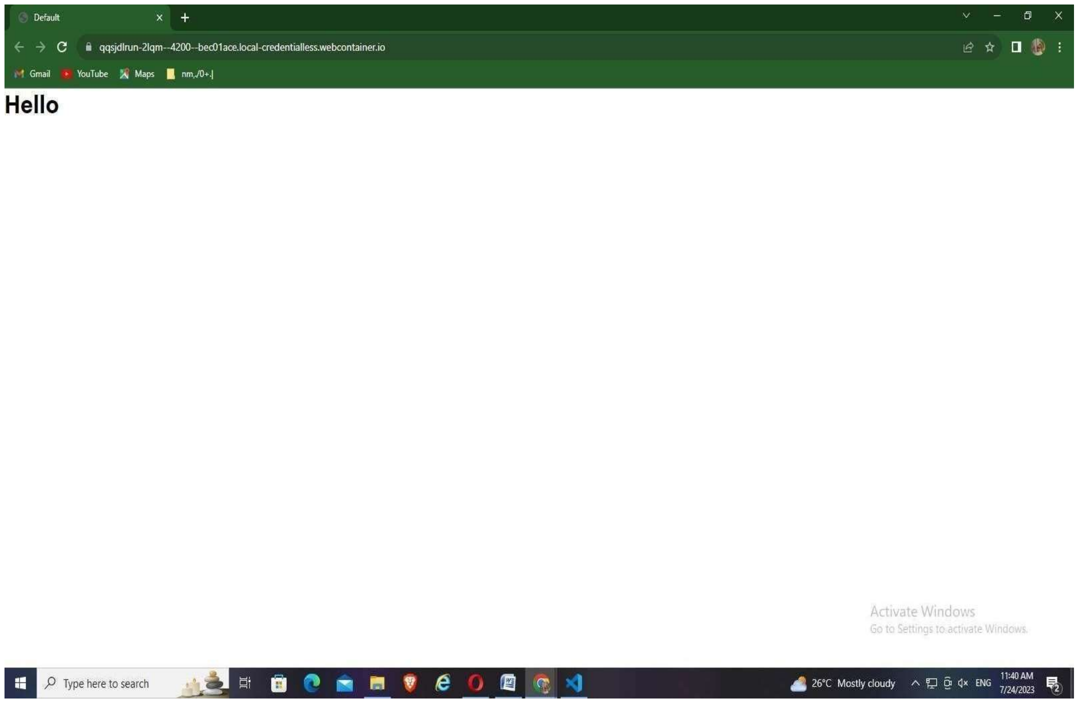
```
:host {
  --content-padding: 10px;
}
header {
  display: block;
  height: 60px;
  padding: var(--content-padding);
  box-shadow: 0px 5px 25px var(--shadow-color);
}
.content {
  padding: var(--content-padding);
}
```

**App.component.ts**

```
import { Component } from '@angular/core';


@Component({
  selector: 'app-root',
  standalone: true,
  imports: [],
  template: `<h1>Hello</h1>`,
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  title = 'default';}
```

**Output:**

```
~/projects/qqsjdl--run

❯ npm install --legacy-peer-deps && npm start

npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher.  Older versions may use Math.random() in certain circumstances, which is known to be problematic.  See https://v8.dev/blog/math-random for details.

npm WARN deprecated har-validator@5.1.5: this library is no longer supported

npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142

npm WARN deprecated protractor@7.0.0: We have news to share - Protractor is deprecated and will reach end-of-life by Summer 2023. To learn more and find out about other options please refer to this post on the Angular blog. Thank you for using and contributing to Protractor. https://goo.gle/state-of-e2e-in-angular



added 1062 packages in 10s



104 packages are looking for funding

  run `npm fund` for details



> angular.io-example@0.0.0 start

> ng serve



✔ Browser application bundle generation complete.



Initial Chunk Files  | Names         | Raw Size

vendor.js            | vendor        |  1.98 MB |
polyfills.js          | polyfills     | 333.17 kB |
styles.css, styles.js | styles        | 231.00 kB |
runtime.js            | runtime       |  6.53 kB |
main.js               | main          |  4.11 kB |

                      | Initial Total |   2.54 MB

Build at: 2023-07-24T06:01:39.536Z - Hash: 12a28dce4fb5ef2b - Time: 6366ms

** Angular Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200/ **

✓ Compiled successfully.
✓ Browser application bundle generation complete.

5 unchanged chunks

Build at: 2023-07-24T06:01:40.019Z - Hash: 12a28dce4fb5ef2b - Time: 377ms

✓ Compiled successfully.
```



```
Hello
```

## Exercise-1(c)

**Course Name:** Angular JS Module.

**Module Name:** Elements of Template Add an event to the hello component template and when it is clicked, it should change the course Name.

**hello.component.ts:**

```
import { Component, OnInit } from '@angular/core';

@Component({

  selector: 'app-hello',

  templateUrl: "./hello.component.html",

  styleUrls: ['./hello.component.css']

})

export class HelloComponent implements OnInit {

  courseName = "MSD";

  constructor() { }

  ngOnInit() {

  }

  changeName() {

    this.courseName = "CSE";

  }

}
```

**hello.component.html:**

```
<h1>Welcome</h1>

<h2>Course Name: {{ courseName }}</h2>
```

```
<button><p (click)="changeName()">Click here to change</p></button>
```

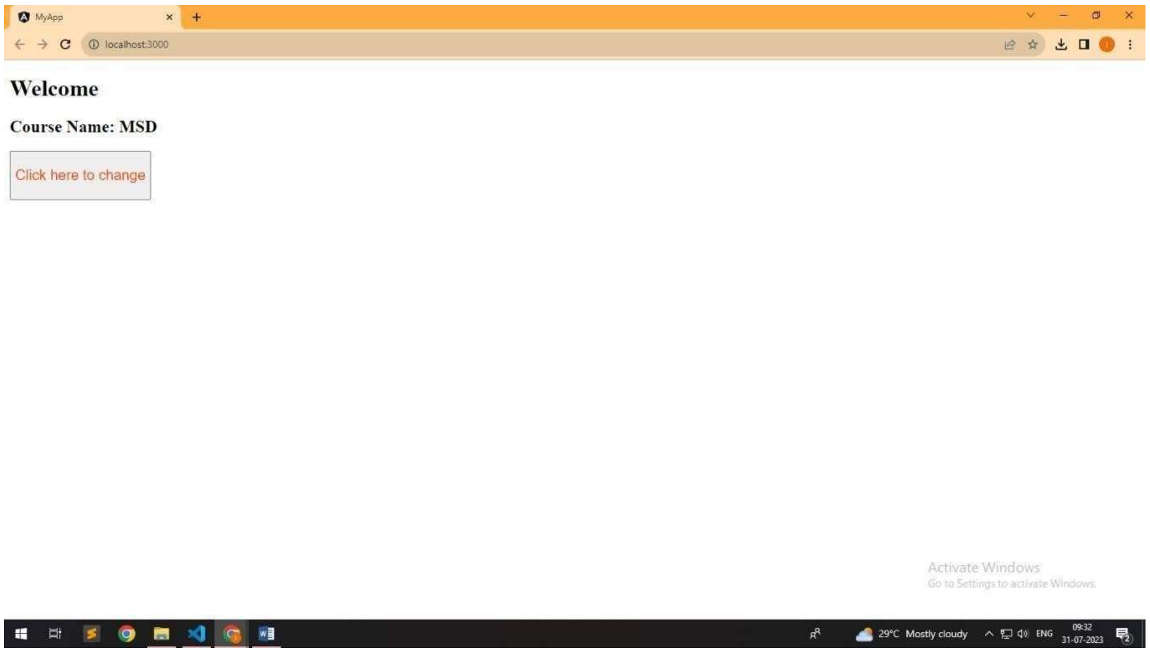
**hello.component.css:**

```
p {
    color:rgb(255, 60, 0);
    font-size:20px;
}
```

**<u>Output:</u>**

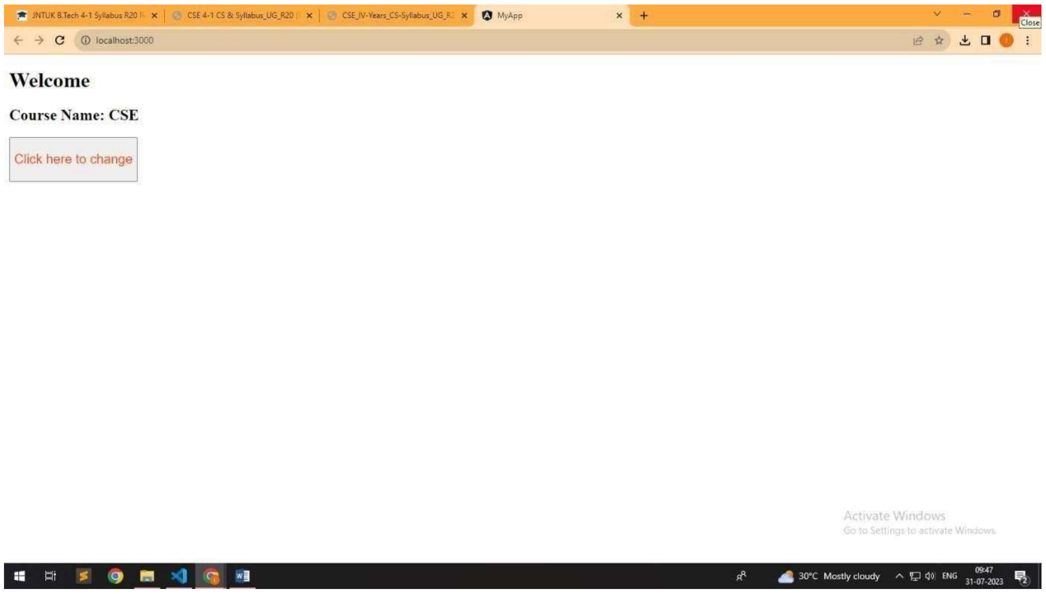
```
PS C:\Users\CSE\Desktop\angular\MyApp\src\app> ng serve --open --port 3000

✓ Browser application bundle generation complete.


Initial Chunk Files  | Names        | Raw Size

vendor.js            | vendor       |  2.28 MB |

polyfills.js         | polyfills    | 333.15 kB |

styles.css, styles.js | styles      | 230.44 kB |

main.js              | main         |  8.79 kB |

runtime.js           | runtime      |  6.51 kB |


                     | Initial Total |   2.85 MB


Build at: 2023-07-31T03:54:06.304Z - Hash: 538059ec4cb5376e - Time: 12617ms


** Angular Live Development Server is listening on localhost:3000, open your browser on
http://localhost:3000/ **
```

√ Compiled successfully.



Welcome

Course Name: MSD



Welcome

Course Name: CSE

## Exercise-2(a)

**Course Name:** Angular JS
**Module Name:** Structural Directives - ngIf
Create a login form with username and password fields. If the user enters the correct credentials, it should render a "Welcome <<username>>" message otherwise it should render "Invalid Login!!! Please try again..." message

- Directives are used to change the behavior of components or elements. It can be used **in the form of HTML attributes**.
- You can create directives using classes attached with @Directive decorator which adds metadata to the class.

### Why Directives?

- It modify the DOM elements
- It creates reusable and independent code
- It is used to create custom elements to implement the required functionality

### Types of Directives

There are three types of directives available in Angular

### Components

- Components are directives with a template or view.
- @Component decorator is actually @Directive with templates

### Structural Directives

- A Structural directive changes the DOM layout by adding and removing DOM elements.

Syntax: *directive-name=expression

- Angular has few built-in structural directives such as:
  - ngIf
  - ngFor
  - ngSwitch

**ngIf:** ngIf directive renders components or elements conditionally based on whether or not an expression is true or false.

**Syntax**:

**1. *ngIf = "expression"**

Open already created app and do the required modifications in **app.component.ts** file

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  isAuthenticated!: boolean;
  submitted = false;
  userName!: string;
  onSubmit(name: string, password: string) {
    this.submitted = true;
    this.userName = name;
    if (name === 'sirisha533' && password === '5363') {
      this.isAuthenticated = true;
    } else {
      this.isAuthenticated = false;
    }
  }
}
```

**app.component.html**

```
<div *ngIf="!submitted">
<form>
  <label>User Name</label>
  <input type="text" #username /><br /><br />
  <label for="password">Password</label>
  <input type="password" name="password" #password /><br />
</form>
<button (click)="onSubmit(username.value,  password.value)">Login</button>
</div>
<div *ngIf="submitted">
<div *ngIf="isAuthenticated; else failureMsg">
  <h4>Welcome {{ userName }}</h4>
</div>
<ng-template #failureMsg>
  <h4>Invalid Login !!! Please try again...</h4>
</ng-template>
<button type="button" (click)="submitted = false">Back</button>
</div>
```

**app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
   bootstrap: [AppComponent],
})export class AppModule {}
```

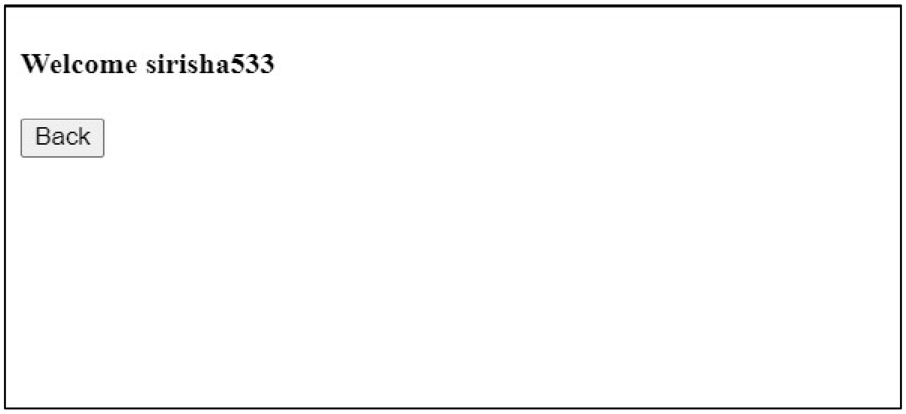
**index.html**

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

- Save the files and the check the output in browser

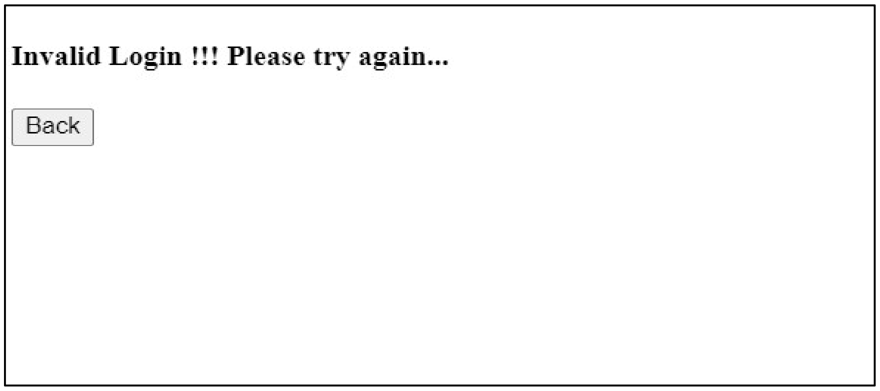
**Output:**

If the usename and password are correct console will display this output.



**Welcome sirisha533**

If the username and password are incorrect **Invalid Login !!** will be displayed.



**Invalid Login !!! Please try again...**

## Exercise-2(b)

**Course Name:** Angular JS
**Module Name:** ngFor
Create a courses array and rendering it in the template using ngFor directive in a list format.

**1. Write the below-given code in app.component.ts**

```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  courses: any[] = [

    { id: 1, name: 'TypeScript' },

    { id: 2, name: 'Angular' },

    { id: 3, name: 'Node JS' },

    { id: 1, name: 'TypeScript' }

  ];

}
```

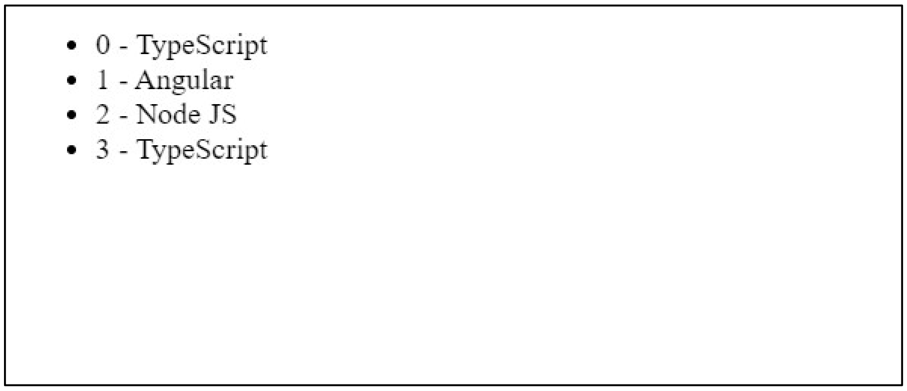
**2. Write the below-given code in app.component.html**

```
<ul>

  <li *ngFor="let course of courses; let i = index">

    {{ i }} - {{ course.name }}

  </li>

</ul>
```

**3. Save the files and check the output in the browser**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  courses: any[] = [
    { id: 1, name: 'TypeScript' },
    { id: 2, name: 'Angular' },
    { id: 3, name: 'Node JS' },
    { id: 1, name: 'TypeScript' }
  ];
}
```

**Output:**



```
• 0 - TypeScript
• 1 - Angular
• 2 - Node JS
• 3 - TypeScript
```

**Exercise-2(c)**

**Course Name:** Angular JS
**Module Name:** ngSwitch
Display the correct option based on the value passed to ngSwitch directive.

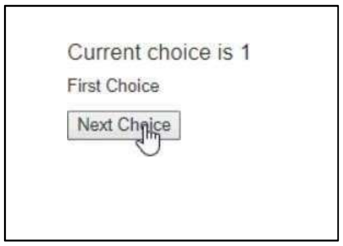
**App.componet.ts**

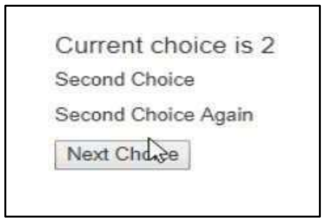
```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css'],

})

export class AppComponent {

  c=0;

  nextChoice(){

    this.c++;

      if(this.c>6){

              this.c=0;} }}
```

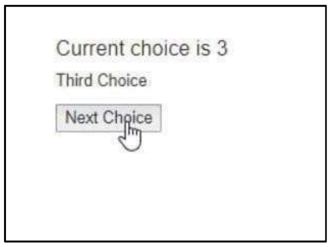
**App.component.html**

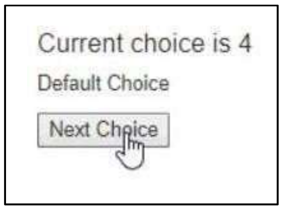
```
<h4>current choice is {{c}}</h4>

<div [ngSwitch]="c">

     <p *ngSwitchCase="0">Start </p>

    <p *ngSwitchCase="1">HTML</p>

    <p *ngSwitchCase="2">CSS</p>

    <p *ngSwitchCase="3">Javascript</p>

    <p *ngSwitchCase="4">Typescript</p>
    <p *ngSwitchCase="5">Angular</p>

    <p *ngSwitchCase="6">Exit</p>

</div>

<div>

    <button (click)="nextChoice()"> Next</button>

</div>
```

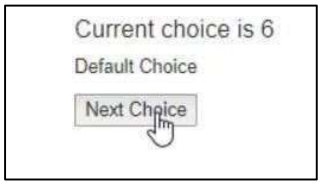
**Output:**


Current choice is 1
First Choice


Current choice is 2
Second Choice
Second Choice Again


Current choice is 3
Third Choice


Current choice is 4
Default Choice


Current choice is 6
Default Choice

In the above output, the current choice is incremented up to 6 and then it is reset to 0

## Exercise-2(d)

**Course Name:** Angular JS
**Module Name:** Custom Structural Directive
Create a custom structural directive called 'repeat' which should repeat the element given a number of times.

**Step-1:**Generate a new directive by using Directive method

```
D:\MyApp> ng generate directive repeat
```

This will create two files under the src\app folder with names repeat.directive.ts and repeat.directive.spec.ts (this is for testing). Now the app folder structure will look as shown below.

**Step-2: app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

import { RepeatDirective } from './repeat.directive';

@NgModule({

  declarations: [

    AppComponent,

    RepeatDirective

  ],

  imports: [

    BrowserModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})
```

```
export class AppModule { }
```

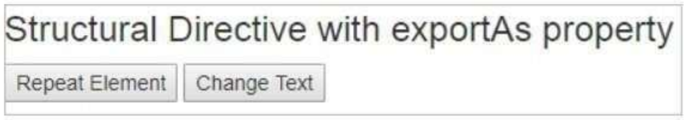
**repeat.directive.ts**

```
import { Directive ,TemplateRef,ViewContainerRef,Input} from '@angular/core';


@Directive({

  selector: '[appRepeat]'

})

export class RepeatDirective {


   constructor(private templatRref:TemplateRef<any>,private viewContainer:ViewContainerRef)
{}

  @Input() set appRepeat(count:number){

    for(let i=0;i<count;i++){

      this.viewContainer.createEmbeddedView(this.templatRref);

    }

  }

}
```

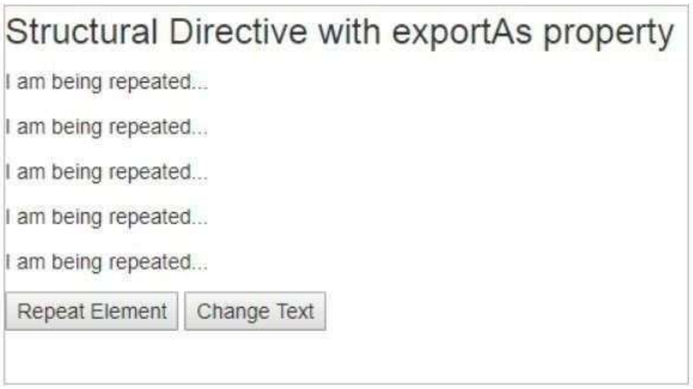
**App.component.html**

```
<h3>Structural Directive with exportAs property</h3>

<ng-template appRepeat #rd="repeat" #ct="changeText">

  <p>I am being repeated...</p>

</ng-template>

<button (click)="rd.repeatElement(5)">Repeat Element</button>

<button (click)="ct.changeElementText(5)">Change Text</button>
```

**Output:**


Structural Directive with exportAs property

When the 'Repeat Element' button is clicked, it renders the below output:


Structural Directive with exportAs property

I am being repeated...

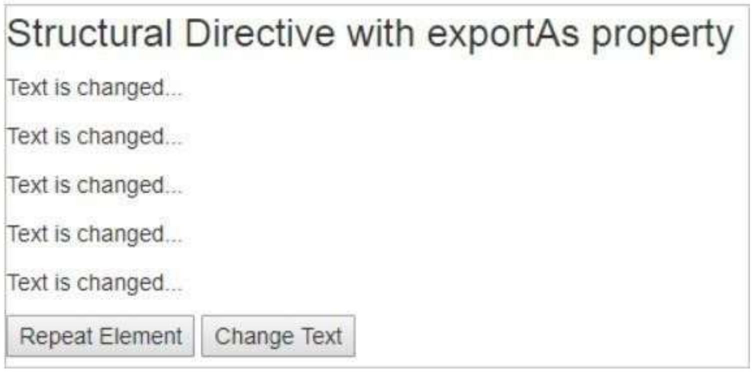
I am being repeated...

I am being repeated...

I am being repeated...

I am being repeated...

Vhen the 'Change Text' button is clicked, it renders the below output:


Structural Directive with exportAs property

Text is changed...

Text is changed...

Text is changed...

Text is changed...

Text is changed...

## Exercise-3(a)

**Module Name:** Attribute directives(ngStyle)

Apply multiple CSS properties to a paragraph in a component using ngStyle.

Attribute directives change the appearance/behavior of a component/element.

Following are built-in attribute directives:

- ngStyle
- ngClass

This directive is used to modify a component/element's style. You can use the following syntax to set a single CSS style to the element which is also known as style binding

```
[style.<cssproperty>] = "value"
```

If there are more than one CSS styles to apply, you can use **ngStyle** attribute.

Now apply the following modifications in the files of MyApp.

**app.component.ts**

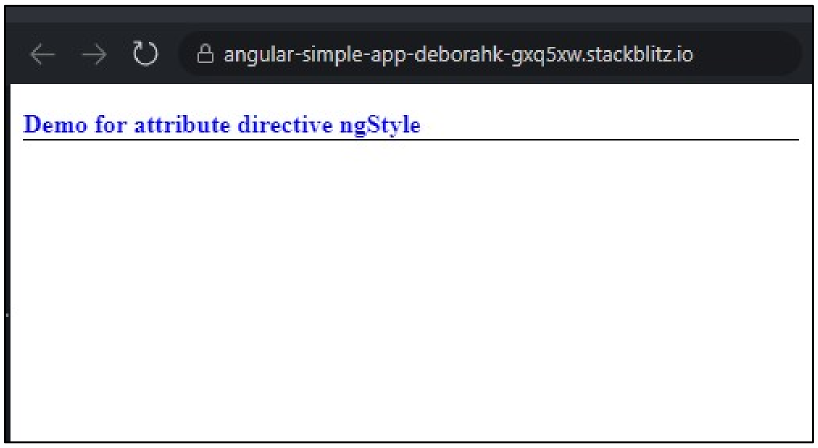
```
import { Component } from '@angular/core';


@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css'],

})

export class AppComponent {

  colorName = 'blue';

  fontWeight = 'bold';
  borderStyle = '1px solid black';}
```

**app.component.html**

```
<p
  [ngStyle]="{
    color: colorName,
    'font-weight': fontWeight,
    borderBottom: borderStyle
  }"
>
  Demo for attribute directive ngStyle
</p>
```

Save the changes to files and check the browser for output.

**Output:**



Demo for attribute directive ngStyle

## Exercise-3(b)

**Module Name:** ngClass

Apply multiple CSS classes to the text using ngClass directive.

It allows you to dynamically set and change the CSS classes for a given DOM element. Use the following syntax to set a single CSS class to the element which is also known as class binding.

[class.<css_class_name>] = "property/value"

If you have **more than one CSS classes to apply**, then you can go for ngClass syntax.

**Syntax:**[ngClass] = "{css_class_name1 : Boolean expression, css_class_name2: Boolean expression, ……}"

1. Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  isBordered = true;}
```

Here we created a Boolean expression **isBordered** to evaluate the border style for html pag

2. Write the below-given code in **app.component.html.**

```
<div [ngClass]="{bordered: isBordered}">

  Border {{ isBordered ? "ON" : "OFF" }}

</div>
```

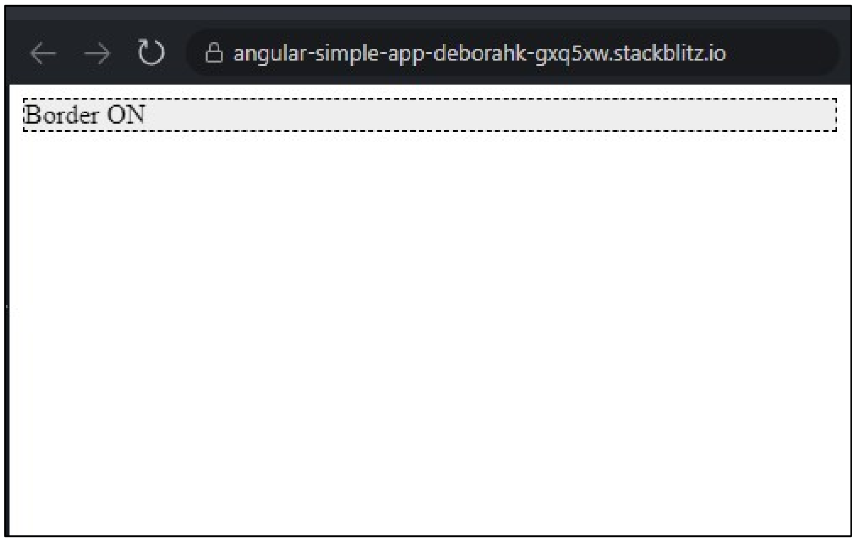
3. In **app.component.css**, add the following CSS class.

```
.bordered {
  border: 1px dashed black;
  background-color: #eee;
}
```

You can

Save the changes and check the browser for output.

**Output:**



```
Border ON
```

## Exercise-3(c)

**Module Name:** Custom attribute directive

Create an attribute directive called 'showMessage' which should display the given message in a paragraph when a user clicks on it and should change the text color to red.

You can create a custom attribute directive when there is no built-in directive available for the required functionality. For Example, consider the following problem statement:

To create a custom attribute directive, we need to create a class annotated with @Directive

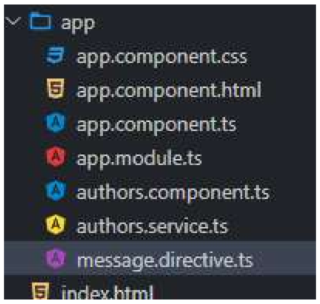
```
@Directive({

})

class MyDirective { }
```

**Example**:

1. Generate a directive called 'message' using the following command

```
D:\MyApp> ng generate directive message
```

This will create two files under the src\app folder with the names message.directive.ts and message.directive.spec.ts (this is for testing). Now the app folder structure will look as shown below:



```
app
  app.component.css
  app.component.html
  app.component.ts
  app.module.ts
  authors.component.ts
  authors.service.ts
  message.directive.ts
index.html
```

It also adds message directive to the root module i.e., **app.module.ts** to make it available to the entire module as shown below

2. Above command will add MessageDirective class to the declarations property in the **app.module.ts** file

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';


import { AppComponent } from './app.component';
import { MessageDirective } from './message.directive';


@NgModule({
 declarations: [
 AppComponent,
 MessageDirective
 ],
 imports: [
  BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

3. Open the **message.directive.ts** file and add the following code

```
  import { Directive, ElementRef, Renderer2, HostListener, Input } from '@angular/core';
      @Directive({
   selector: '[appMessage]',
  })
  export class MessageDirective {
   @Input('appMessage') message!: string;
```

```
      constructor(private el: ElementRef, private renderer: Renderer2) {
   renderer.setStyle(el.nativeElement, 'cursor', 'pointer');}
   @HostListener('click') onClick() {
   this.el.nativeElement.innerHTML = this.message;
   this.renderer.setStyle(this.el.nativeElement, 'color', 'red');
   }
  }
```

4. Write the below-given code in **app.component.html**

```
<h3>Attribute Directive</h3>
<p [appMessage]="myMessage">Click Here</p>
```

5. Add the following CSS styles to the **app.component.css** file

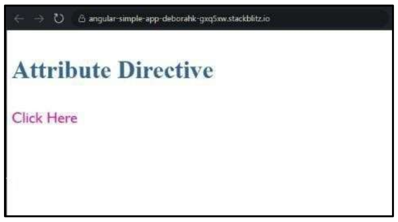
```
h3 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
p {
  color: #ff0080;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 150%;
}
```

6. Add the following code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  myMessage = 'Hello, I am from attribute directive';
}
```

Save the changes and check the browser for output.

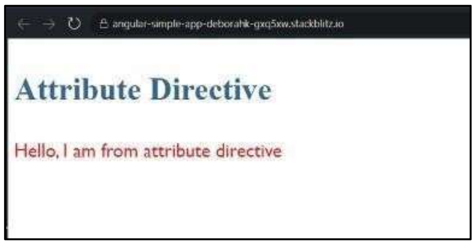
**Output:**


angular-simple-app-deborahk-gxq5xw.stackblitz.io

Attribute Directive

Click Here

$\longrightarrow$


angular-simple-app-deborahk-gxq5xw.stackblitz.io

Attribute Directive

Hello, I am from attribute directive

## Exercise-4(a)

**Course Name:** Angular JS

**Module Name:** Property Binding .(Binding image with class property using property binding.)

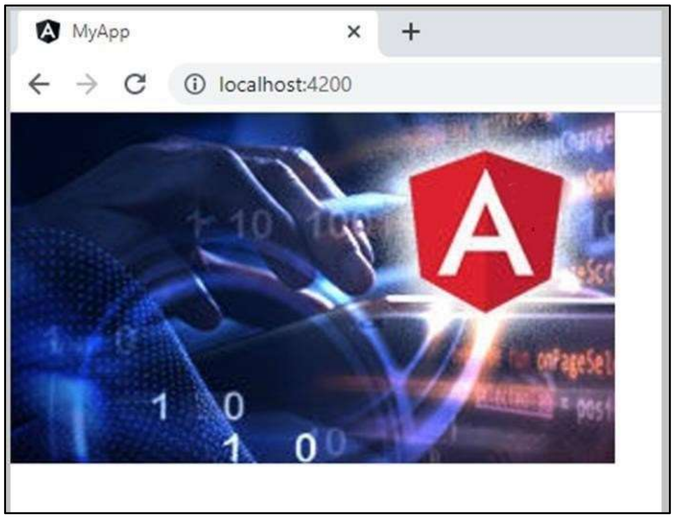
**Write the following code in app.component.ts as shown below**

```
import { Component } from

'@angular/core'; @Component({

  selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class

AppComponent { imgUrl =

'assets/imgs/logo.png';

}
```

Create a folder named "imgs" inside src/assets and place a logo.png file inside it. Write the following code in app.component.html as shown below

**<img [src]='imgUrl'>**

**Output:**



**Exercise-4(b)**

**Module Name:** Attribute Binding

Binding colspan attribute of a table element to the class property to display the following output

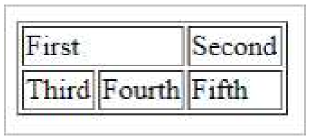
Write the below-given code in **app.component.ts**

```
import { Component } from
'@angular/core'; @Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class
AppComponent {
colspanValue = '2';
}
```

Write the below-given code in **app.component.html**

```
<table border=1>
<tr>
<td [attr.colspan]="colspanValue"> First </td>
<td>Second</td>
</tr><tr>
<td>Third</td>
<td>Fourth</td>
<td>Fifth</td>
</tr></table>
```

**OUTPUT:-**



| | | |
|---|---|---|
| First | | Second |
| Third | Fourth | Fifth |

**Exercise-4(c)**

**Module Name:** Style and Event Binding

**Problem Statement :** Binding an element using inline style and user actions like entering text in input fields.

Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
 name = 'Angular';
}
```

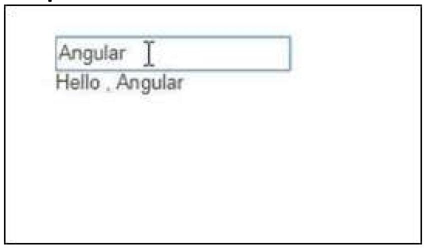
Write the below-given code in **app.component.html**

```
<input type="text" [(ngModel)]="name"><br/>
<div>Hello , {{ name }}</div>
```

Write the below-given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from
'@angular/forms'; import { AppComponent }
from './app.component'; @NgModule({
  declarations: [
AppComponent
  ],
  imports: [
BrowserModule,
FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

**Output:**



```
Angular
Hello , Angular
```

**Exercise-5(a)**

**Module Name:** Built in Pipes .Displaying the product code in lowercase and product name in uppercase using built-in pipes. The output is as shown below

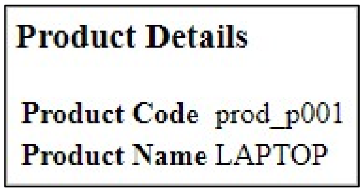
Write the below-given code in **app.component.ts**

```
import { Component } from
'@angular/core'; @Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class
AppComponent { title =
'product details';
productCode =
'PROD_P001';
productName = 'Laptop';
}
```

Write the below-given code in **app.component.html**

```
<h3>{{ title | titlecase}} </h3>
<table style="text-align:left">
    <tr>
            <th> Product Code </th>
            <td>{{ productCode | lowercase }} </td>
    </tr>
    <tr>
            <th> Product Name </th>
            <td>{{ productName | uppercase }} </td>
    </tr>
</table>
```

**OUTPUT:-**



```
Product Details

Product Code  prod_p001
Product Name  LAPTOP
```

**Excícisc-5(b)**

**Module Name:** Passing Parameters to Pipes

Applying built-in pipes with parameters to display product details. The output is as shown below

We have applied currency pipe to product price with locale setting as 'fr' i.e., French. According to the French locale, the currency symbol will be displayed at the end of the price as shown in the above output.

Write the below-given code in **app.component.ts**

```
import { Component } from
'@angular/core'; @Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class
AppComponent { title =
'product details';
productCode =
'PROD_P001';
productName = 'Apple MPTT2 MacBook
Pro'; productPrice = 217021;
purchaseDate =
'1/17/2018'; productTax
= '0.1';
productRating = 4.92;
}
```

Write the below-given code in **app.component.html**

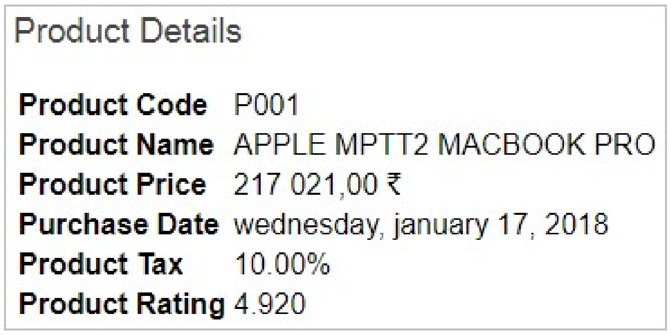
```
<h3>{{ title | titlecase}} </h3>
<table style="text-align:left">
<tr>
<th> Product Code </th>
<td>{{ productCode | slice:5:9 }} </td>
</tr>
<tr>
<th> Product Name </th>
<td>{{ productName | uppercase }} </td>
</tr>
<tr>
<th> Product Price </th>
<td>{{ productPrice | currency: 'INR':'symbol':'':'fr' }} </td>
</tr>
<tr>
<th> Purchase Date </th>
<td>{{ purchaseDate | date:'fullDate' | lowercase}} </td>
</tr>
```

```
<tr>
<th> Product Tax </th>
<td>{{ productTax | percent : '.2' }} </td>
</tr>
<tr>
<th> Product Rating </th>
    <td>{{ productRating | number:'1.3-5'}} </td>
</tr>
</table>
```

Write the below-given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { registerLocaleData } from
'@angular/common'; import localeFrench from
'@angular/common/locales/fr';
registerLocaleData(localeFrench);
@NgMod
ule({
declaratio
ns:        [
AppCom
ponent
  ],
  imports: [
BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule{ }
```

**OUI'PUI':-**



```
Product Details

Product Code     P001
Product Name     APPLE MPTT2 MACBOOK PRO
Product Price    217 021,00 ₹
Purchase Date    wednesday, january 17, 2018
Product Tax      10.00%
Product Rating   4.920
```

**Excícisc-5(c)**

**Module Name:** Passing Parameters to Pipes

Loading Courses list Component in the root component when a user clicks on the View courses list button as shown below

1. Create a component called coursesList using the following CLI command

```
D:\MyApp>ng generate component coursesList
```

The above command will create a folder with name courses-list with the following files

- courses-list.component.ts
- courses-list.component.html
- courses-list.component.css
- courses-list.component.spec.ts

2. CoursesListComponent class will be added in the **app.module.ts** file

```
import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

import { CoursesListComponent } from './courses-list/courses-list.component';

@NgModule({

  declarations: [

AppComponent,

CoursesListCompo

nent

],  imports: [

BrowserModule

],  providers: [],

  bootstrap: [AppComponent]

})export class AppModule { }
```

3. Write the below-given code in **courses-list.component.ts**

```
import { Component, OnInit } from

'@angular/core'; @Component({

  selector: 'app-courses-list',

templateUrl: './courses-list.component.html',
styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {
  courses = [
{ courseId: 1, courseName: "Node JS" },
{ courseId: 2, courseName: "Typescript" },
{ courseId: 3, courseName: "Angular" },
{ courseId: 4, courseName: "React JS" }
  ];
}
```

4. Write the below-given code in **courses-list.component.html**

```
<table border="1">
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
</tr>
</thead>
<tbody>
<tr *ngFor="let course of courses">
<td>{{ course.courseId }}</td>
<td>{{ course.courseName }}</td>
</tr>
</tbody>
</table>
```

5. Add the following code in **courses-list.component.css**

```
tr{
text-align:center;
}
```

6. Write the below-given code in **app.component.html**

```
<h2> Popular Courses</h2>
<button (click)="show = true">View Courses list</button><br /><br />
```

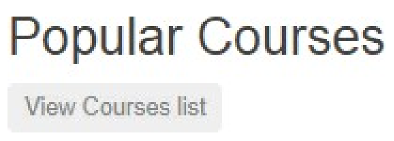
```
<div *ngIf="show">

<app-courses-list></app-courses-list>

</div>
```

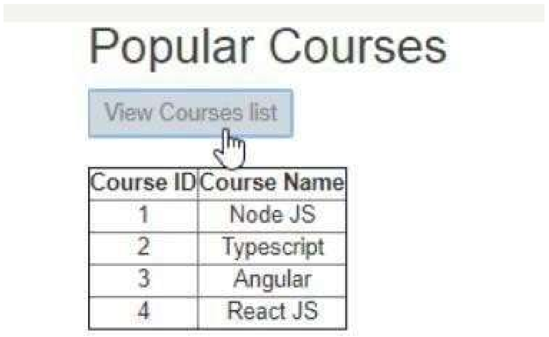
7. Write the below-given code in **app.component.ts**

```
import { Component } from
'@angular/core'; @Component({

        selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class
AppComponent {
show!:boolean;
}
```

8. Save the files and check the output in the browser

**OUTPUT:-**





| Course ID | Course Name |
|---|---|
| 1 | Node JS |
| 2 | Typescript |
| 3 | Angular |
| 4 | React JS |

## Exercise-6(a)

**Module Name**: Passing data from Container Component to Child Component

Create an AppComponent that displays a dropdown with a list of courses as values in it. Create another component called the CoursesList component and load it in AppComponent which should display the course details. When the user selects a course from the.

**Steps:**

1. Open the courses-list.component.ts file created in the example of nested components and add the following code

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'app-courses-list',
  templateUrl: './courses-list.component.html',
  styleUrls: ['./courses-list.component.css'],
})
export class CoursesListComponent {
  courses = [
    { courseId: 1, courseName: 'Node JS' },
    { courseId: 2, courseName: 'Typescript' },
    { courseId: 3, courseName: 'Angular' },
    { courseId: 4, courseName: 'React JS' },
  ];
  course!: any[];
  @Input() set cName(name: string) {
    this.course = [];
    for (var i = 0; i < this.courses.length; i++) {
      if (this.courses[i].courseName === name) {
      this.course.push(this.courses[i]);
      }
    }
  }
}
```

2. Open courses-list.component.html and add the following code

```
<table border="1" *ngIf="course.length > 0">
  <thead>
    <tr>
      <th>Course ID</th>
      <th>Course Name</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let c of course">
      <td>{{ c.courseId }}</td>
      <td>{{ c.courseName }}</td>
    </tr>
  </tbody>
</table>
```

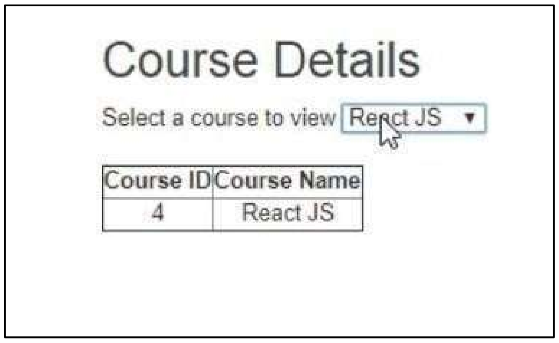
3. Add the following in app.component.html

```
<h2>Course Details</h2>
Select a course to view
<select #course (change)="name = course.value">
  <option value="Node JS">Node JS</option>
  <option value="Typescript">Typescript</option>
  <option value="Angular">Angular</option>
  <option value="React JS">React JS</option></select><br /><br />
<app-courses-list  [cName]="name"></app-courses-list>
```

4. Add the following in app.component.ts

```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

  styleUrls: ['./app.component.css'],

  templateUrl: './app.component.html'

})

export class AppComponent {

  name!: string;

}
```

**Output:**



Course Details

Select a course to view React JS ▼

| Course ID | Course Name |
|---|---|
| 4 | React JS |

## Exercise-6(b)

**Module Name**: Passing data from Child Component to ContainerComponent

Create an AppComponent that loads another component called the CoursesList component. Create another component called CoursesListComponent which should display the courses list in a table along with a register .button in each row. When a user clicks on the.

Steps:

1. Open the courses-list.component.ts file created in the previous example and add the following code

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';

@Component({

  selector: 'app-courses-list',

  templateUrl: './courses-list.component.html',

  styleUrls: ['./courses-list.component.css']

})

export class CoursesListComponent {

  @Output() registerEvent = new EventEmitter<string>();

  courses = [

    { courseId: 1, courseName: 'Node JS' },

    { courseId: 2, courseName: 'Typescript' },

    { courseId: 3, courseName: 'Angular' },

    { courseId: 4, courseName: 'React JS' }

  ];

  register(courseName: string) {

    this.registerEvent.emit(courseName);

  }}
```

2. Open courses-list.component.html and add the following code

```
<table border="1">
  <thead>
    <tr>
      <th>Course ID</th>
      <th>Course Name</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let course of courses">
      <td>{{ course.courseId }}</td>
      <td>{{ course.courseName }}</td>
      <td><button (click)="register(course.courseName)">Register</button></td>
    </tr>
  </tbody>
</table>
```

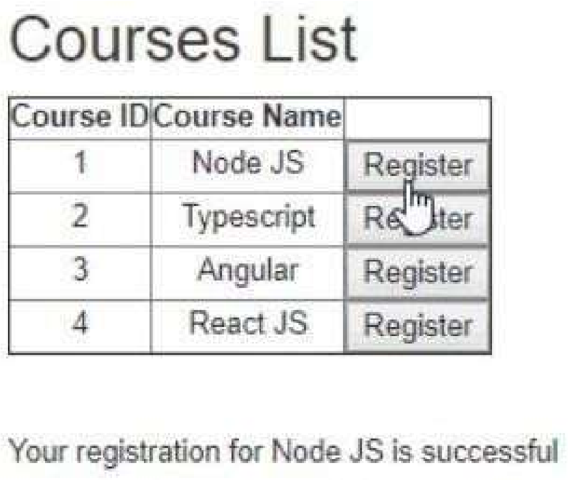
3. Add the following in app.component.html

```
<h2>Courses List</h2>
<app-courses-list  (registerEvent)="courseReg($event)"></app-courses-list>
<br /><br />
<div *ngIf="message">{{ message }}</div>
```

4. Add the following code in app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  message!: string;
  courseReg(courseName: string) {
    this.message = `Your registration for ${courseName} is successful`;
  }
}
```

Output:



Courses List

| Course ID | Course Name | |
|---|---|---|
| 1 | Node JS | Register |
| 2 | Typescript | Register |
| 3 | Angular | Register |
| 4 | React JS | Register |

Your registration for Node JS is successful

## Exercise-6(c)

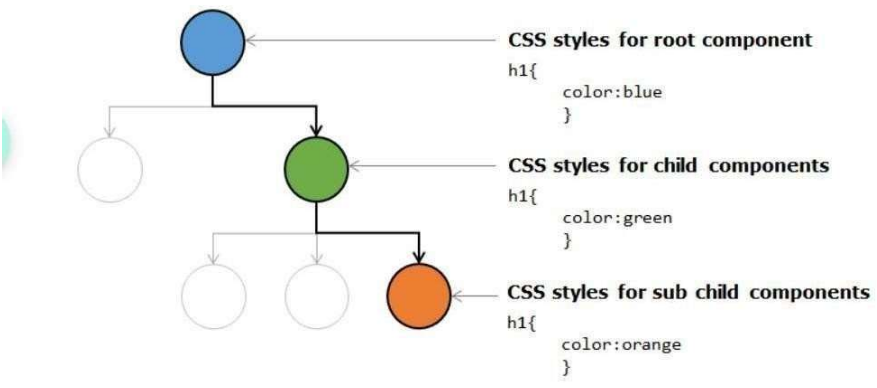
**Module Name:** Shadow DOM

Apply ShadowDOM and None encapsulation modes to components.

Shadow DOM is a web components standard by W3C. It **enables encapsulation for DOM tree and styles**. Shadow DOM hides DOM logic behind other elements and confines styles only for that component.

For example, in an Angular application, n number of components will be created and each component will have its own set of data and CSS styles. When these are integrated, there is a chance that the data and styles may be applied to the entire application. Shadow DOM encapsulates data and styles for each component to not flow through the entire application

In the below example shown, each component is having its own styles defined and they are confined to themselves:


CSS styles for root component
h1{
color:blue
}
CSS styles for child components
h1{
color:green
}
CSS styles for sub child components
h1{
color:orange
}

Steps:

1. Create a component called **First** using the following CLI command

```
1.   D:\MyApp>ng generate component first
```

2. Write the below-given code in **first.component.css**

```
1.   .cmp {
2.     padding: 6px;
```

```
  margin: 6px;
  border: blue 2px solid;
}
```

3. Write the below-given code in **first.component.html**

```
<div class="cmp">First Component</div>
```

4. Create a component called **Second** using the following CLI command

```
D:\MyApp>ng generate component second
```

5. Write the below-given code in **second.component.css**

```
.cmp {
    border: green 2px solid;
    padding: 6px;
    margin: 6px;
}
```

6. Write the below-given code in **second.component.html**

```
<div class="cmp">Second Component</div>
```

7. Write the below-given code in **second.component.ts**

```
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-second',
  templateUrl: './second.component.html',
  styleUrls: ['./second.component.css'],
  encapsulation: ViewEncapsulation.ShadowDom
})
export class SecondComponent {

}
```

8. Write the below-given code in **app.component.css**

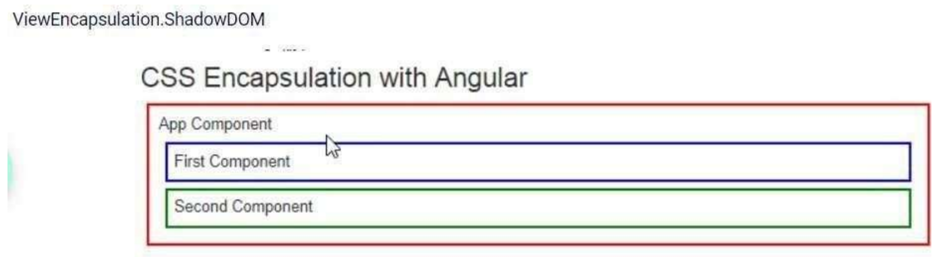
```
.cmp {
  padding: 8px;
  margin: 6px;
  border: 2px solid red;
}
```

9. Write the below-given code in **app.component.html**

```
<h3>CSS Encapsulation with Angular</h3>
<div class="cmp">
    App Component
    <app-first></app-first>
    <app-second></app-second>

</div>
```

10. Save the files and check the output in the browser

**Output:**



```
ViewEncapsulation.ShadowDOM
CSS Encapsulation with Angular
App Component
First Component
Second Component
```

**ViewEncapsulation.None**

1. Set ViewEncapsulation to none mode in **app.component.ts** file

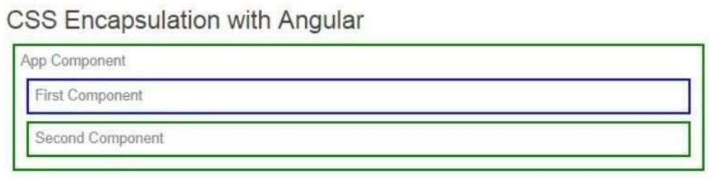
```
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html',
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {}
```

2. Set ViewEncapsulation to none mode in **second.component.ts** file

```
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-second',
  templateUrl: './second.component.html',
  styleUrls: ['./second.component.css'],
  encapsulation: ViewEncapsulation.None
 })
export class SecondComponent {

}
```

2. Save the files and check the output in the browser

**Output:**



```
CSS Encapsulation with Angular
App Component
First Component
Second Component
```

## Exercise-6(d)

**Module Name**: Component Life Cycle

Override component life-cycle hooks and logging the corresponding messages to understand the flow.

Steps:

1. Write the below-given code in app.component.ts

```
import {
    Component, OnInit, DoCheck, AfterContentInit, AfterContentChecked,
    AfterViewInit, AfterViewChecked,
    OnDestroy
} from '@angular/core';
@Component({
    selector: 'app-root',
    styleUrls: ['./app.component.css'],
    templateUrl: './app.component.html'
})
export class AppComponent implements OnInit, DoCheck,
    AfterContentInit, AfterContentChecked,
    AfterViewInit, AfterViewChecked,
    OnDestroy {
    data = 'Angular';
    ngOnInit() {
        console.log('Init');
    }
    ngDoCheck(): void {
        console.log('Change detected');
    }
    ngAfterContentInit(): void {
        console.log('After content init');
    }
    ngAfterContentChecked(): void {
        console.log('After content checked');
    }
    ngAfterViewInit(): void {
        console.log('After view init');
    }
    ngAfterViewChecked(): void {
        console.log('After view checked');
    }
    ngOnDestroy(): void {
        console.log('Destroy');
    }
}
```

2. Write the below-given code in app.component.html

```
<div>
  <h1>I'm a container component</h1>
  <input type="text" [(ngModel)]="data" />
  <app-child  [title]="data"></app-child>
</div>

```

3. Write the below-given code in child.component.ts

```
import { Component, OnChanges, Input } from '@angular/core';
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnChanges {
  @Input() title!: string;
  ngOnChanges(changes: any): void {
    console.log('changes in child:' + JSON.stringify(changes));
  }

}
```

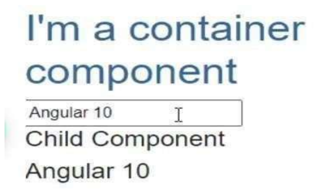
4. Write the below-given code in child.component.html

```
<h2>Child Component</h2>

<h2>{{title}}</h2>
```

5. Ensure FormsModule is present in the imports section of the AppModule.

6. Save the files and check the output in the browser

**Output:**

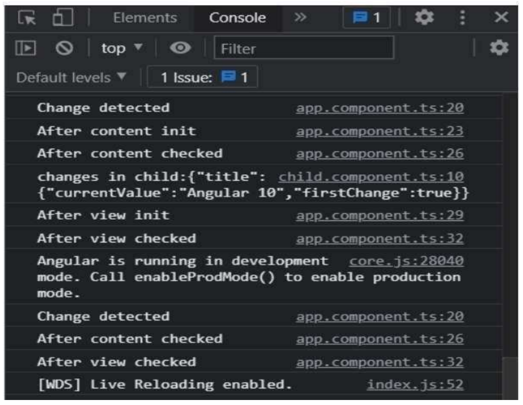


I'm a container component

Angular 10

Child Component

Angular 10



```
Change detected                                  app.component.ts:20
After content init                               app.component.ts:23
After content checked                            app.component.ts:26
changes in child:{"title":                       child.component.ts:10
{"currentValue":"Angular 10","firstChange":true}}
After view init                                  app.component.ts:29
After view checked                               app.component.ts:32
Angular is running in development                core.js:28040
mode. Call enableProdMode() to enable production
mode.
Change detected                                  app.component.ts:20
After content checked                            app.component.ts:26
After view checked                               app.component.ts:32
[WDS] Live Reloading enabled.                    index.js:52
```

## Exercise-7(a)

**Module Name**: Template Driven Forms

Create a course registration form as a template-driven form.

Steps:

1. Create **course.ts** file under the course-form folder and add the following code

```
export class Course {
  constructor(
    public courseId: number,
    public courseName: string,
    public duration: string
  ) { }
}
```

2. Add the following code in the **course-form.component.ts** file

```
import { Component } from '@angular/core';
import { Course } from './course';
@Component({
  selector: 'app-course-form',
  templateUrl: './course-form.component.html',
  styleUrls: ['./course-form.component.css']
})
export class CourseFormComponent {
  course = new Course(1, 'Angular', '5 days');
  submitted = false;
  onSubmit() { this.submitted = true; }

}
```

3. Install **bootstrap**

```
D:\MyApp>npm install bootstrap@3.3.7 --save
```

4. Include boostrap.min.css file in **angular.json** file as shown below

```
 ...
"styles": [
        "styles.css",
        "./node_modules/bootstrap/dist/css/bootstrap.min.css"
      ],
...
```

5. Write the below-given code in **course-form.component.html**

```
<div class="container">
  <div [hidden]="submitted">
    <h1>Course Form</h1>
    <form (ngSubmit)="onSubmit()"  #courseForm="ngForm">
      <div class="form-group">
        <label for="id">Course Id</label>
        <input type="text" class="form-control" required [(ngModel)]="course.courseId" name="id" #id="ngModel">
        <div [hidden]="id.valid || id.pristine" class="alert alert-danger">
          Course Id is required
        </div>
      </div>
      <div class="form-group">
        <label for="name">Course Name</label>
        <input type="text" class="form-control" required [(ngModel)]="course.courseName" name="name" #name="ngModel">
        <div [hidden]="name.valid || name.pristine" class="alert alert-danger">
          Course Name is required
        </div>
      </div>
      <div class="form-group">
        <label for="duration">Course Duration</label>
        <input type="text" class="form-control" required [(ngModel)]="course.duration" name="duration" #duration="ngModel">
        <div [hidden]="duration.valid || duration.pristine" class="alert alert-danger">
          Duration is required
        </div>
      </div>
      <button type="submit" class="btn btn-default" [disabled]="!courseForm.form.valid">Submit</button>
      <button type="button" class="btn btn-default" (click)="courseForm.reset()">New Course</button>
    </form>
  </div>
  <div [hidden]="!submitted">
    <h2>You submitted the following:</h2>
    <div class="row">
      <div class="col-xs-3">Course ID</div>
      <div class="col-xs-9  pull-left">{{ course.courseId }}</div>
    </div>
    <div class="row">
      <div class="col-xs-3">Course Name</div>
      <div class="col-xs-9 pull-left">{{ course.courseName }}</div>
    </div>
    <div class="row">
      <div class="col-xs-3">Duration</div>
      <div class="col-xs-9 pull-left">{{ course.duration }}</div>
    </div>
    <br>
    <button class="btn btn-default" (click)="submitted=false">Edit</button>
  </div>

</div>
```

6. Write the below-given code in **course-form.component.css**

```
input.ng-valid[required] {
  border-left: 5px solid #42A948; /* green */
}
input.ng-dirty.ng-invalid:not(form) {
  border-left: 5px solid #a94442; /* red */

}
```

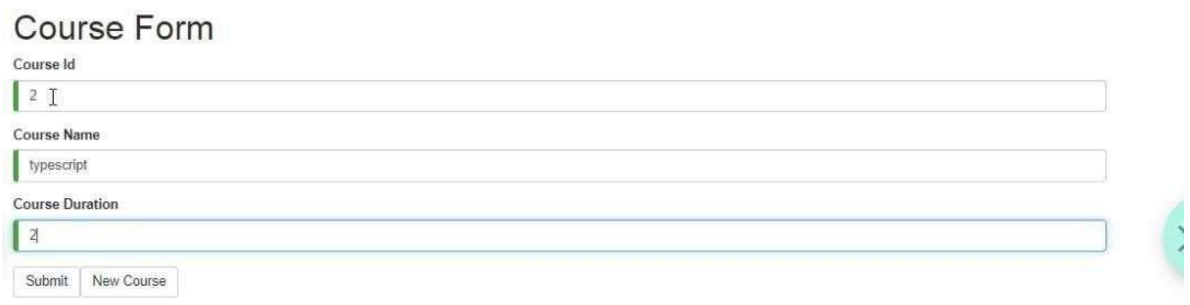
7. Write the below-given code in **app.component.html**

```
<app-course-form></app-course-form>
```

8. Remember to import FormsModule in **app.module.ts.**

9. Save the files and check the output in the browser

**Output:**



Course Form

Course Id

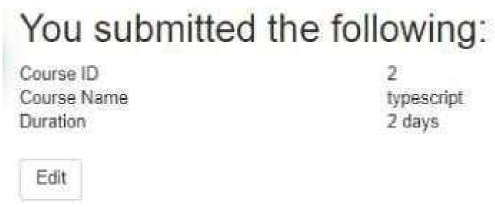
2

Course Name

typescript

Course Duration

2



You submitted the following:

```
Course ID          2
Course Name        typescript
Duration           2 days
```

## Exercise-7(b)

**Module Name**: Model Driven Forms or Reactive Forms

Create an employee registration form as a reactive form.

Steps:

1. Write the below-given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { RegistrationFormComponent } from './registration-form/registration-form.component';

@NgModule({
  declarations: [
    AppComponent,
    RegistrationFormComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2. Create a component called **RegistrationForm** using the following CLI command

```
ng generate component RegistrationForm
```

3. Add the following code in the **registration-form.component.ts** file

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-registration-form',
  templateUrl: './registration-form.component.html',
  styleUrls: ['./registration-form.component.css']
})
export class RegistrationFormComponent implements OnInit {

  registerForm!: FormGroup;
  submitted!:boolean;

  constructor(private formBuilder: FormBuilder) { }

  ngOnInit() {
    this.registerForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      address: this.formBuilder.group({
        street: [],
        zip: [],
        city: []
      })
    });
  }

}
```

4. Write the below-given code in **registration-form.component.html**

```
<div class="container">
    <h1>Registration Form</h1>
    <form [formGroup]="registerForm">
      <div class="form-group">
        <label>First Name</label>
        <input type="text" class="form-control" formControlName="firstName">
          <div *ngIf="registerForm.controls['firstName'].errors" class="alert alert-danger">
          Firstname field is invalid.
          <p *ngIf="registerForm.controls['firstName'].errors?.['required']">
            This field is required!
          </p>
        </div>
      </div>
      <div class="form-group">
        <label>Last Name</label>
        <input type="text" class="form-control"  formControlName="lastName">
       <div *ngIf="registerForm.controls['lastName'].errors" class="alert alert-danger">
          Lastname field is invalid.
          <p *ngIf="registerForm.controls['lastName'].errors?.['required']">
            This field is required!
          </p>
        </div>
      </div>
      <div class="form-group">
        <fieldset formGroupName="address">
          <legend>Address:</legend>
          <label>Street</label>
          <input type="text" class="form-control"  formControlName="street">
          <label>Zip</label>
          <input type="text" class="form-control"  formControlName="zip">
          <label>City</label>
          <input type="text" class="form-control"  formControlName="city">
        </fieldset>
      </div>
      <button type="submit" class="btn btn-primary" (click)="submitted=true">Submit</button>
    </form>
  <br/>
    <div [hidden]="!submitted">
      <h3> Employee Details </h3>
      <p>First Name: {{ registerForm.get('firstName')?.value }} </p>
      <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
      <p> Street: {{ registerForm.get('address.street')?.value }}</p>
      <p> Zip: {{ registerForm.get('address.zip')?.value }} </p>
      <p> City: {{ registerForm.get('address.city')?.value }}</p>
    </div>
  </div>
```

5. Write the below-given code in **registration-form.component.css**

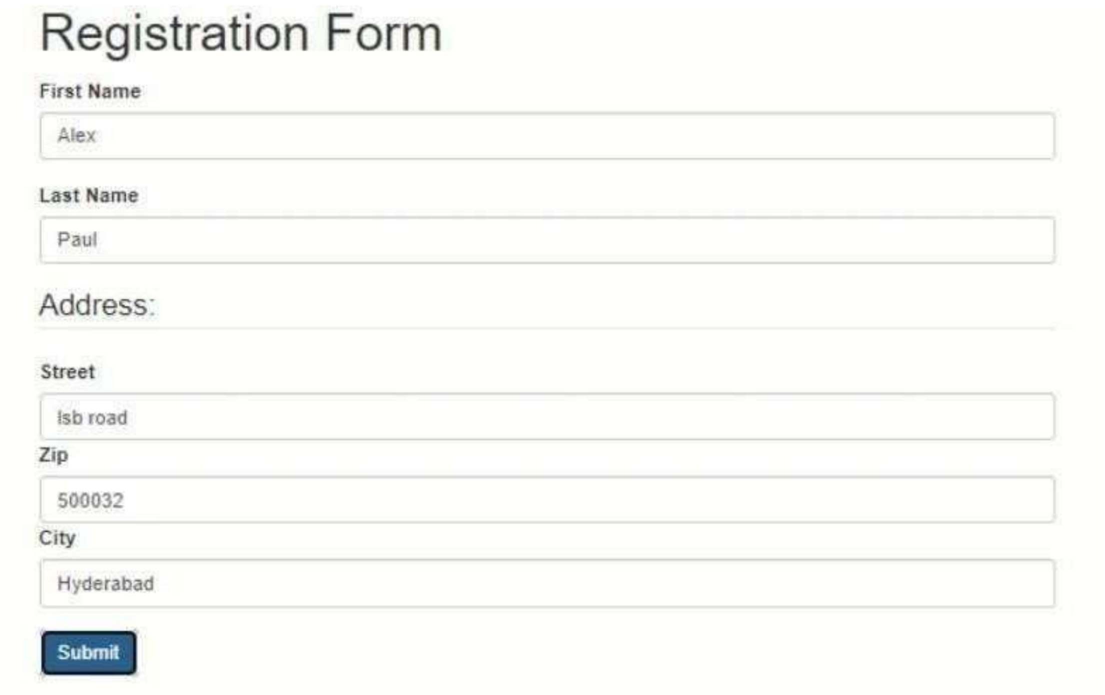
```
.ng-valid[required] {
  border-left: 5px solid #42A948; /* green */
}
.ng-invalid:not(form) {
  border-left: 5px solid #a94442; /* red */
}
```

6. Write the below-given code in **app.component.html**

```
1. <app-registration-form></app-registration-form>
```

7. Save the files and check the output in the browser.

**Output:**



Registration Form

First Name
Alex

Last Name
Paul

Address:

Street
lsb road

Zip
500032

City
Hyderabad

## Exercise-7(c)

**Module Name**: Custom Validators in Reactive Forms

Create a custom validator for an email field in the employee registration form ( reactive form)

Steps:

1. Write a separate function in **registration-form.component.ts** for custom validation as shown below.

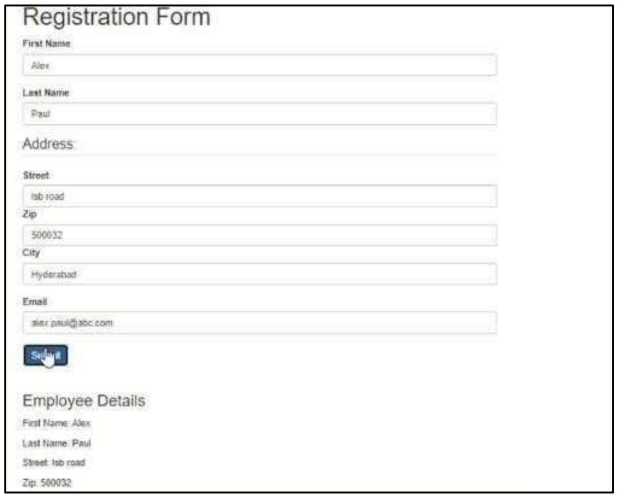
```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-registration-form',
  templateUrl: './registration-form.component.html',
  styleUrls: ['./registration-form.component.css']
})
export class RegistrationFormComponent implements OnInit {

  registerForm!: FormGroup;
  submitted!:boolean;
  constructor(private formBuilder: FormBuilder) { }
  ngOnInit() {
    this.registerForm = this.formBuilder.group({
      firstName: ['',Validators.required],
      lastName: ['', Validators.required],
      address: this.formBuilder.group({
        street: [],
        zip: [],
        city: []
      }),
      email: ['',[Validators.required,validateEmail]]
    });
  }

}
function validateEmail(c: FormControl): any {
  let EMAIL_REGEXP = /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/;

  return EMAIL_REGEXP.test(c.value) ? null : {
    emailInvalid: {
      message: "Invalid Format!"
    }
  };
}
```

2. Add HTML controls for the email field in the **registration-form.component.html** file as shown below

```
<div class="container">
  <h1>Registration Form</h1>
  <form [formGroup]="registerForm">
    <div class="form-group">
      <label>First Name</label>
      <input type="text" class="form-control" formControlName="firstName">
        <div *ngIf="registerForm.controls['firstName'].errors" class="alert alert-danger">
          Firstname field is invalid.
          <p *ngIf="registerForm.controls['firstName'].errors?.['required']">
            This field is required!
          </p>
      </div>
    </div>
    <div class="form-group">
      <label>Last Name</label>
      <input type="text" class="form-control" formControlName="lastName">
      <div *ngIf="registerForm.controls['lastName'].errors" class="alert alert-danger">
          Lastname field is invalid.
          <p *ngIf="registerForm.controls['lastName'].errors?.['required']">
            This field is required!
          </p>
      </div>
    </div>
    <div class="form-group">
      <fieldset formGroupName="address">
        <legend>Address:</legend>
        <label>Street</label>
        <input type="text" class="form-control" formControlName="street">
        <label>Zip</label>
        <input type="text" class="form-control" formControlName="zip">
        <label>City</label>
        <input type="text" class="form-control" formControlName="city">
      </fieldset>
    </div>
    <div class="form-group">
      <label>Email</label>
      <input type="text" class="form-control" formControlName="email" />
      <div *ngIf="registerForm.controls['email'].errors" class="alert alert-danger">
        Email field is invalid.
        <p *ngIf="registerForm.controls['email'].errors?.['required']">
          This field is required!
        </p>
        <p *ngIf="registerForm.controls['email'].errors?.['emailInvalid']">
          {{ registerForm.controls['email'].errors?.['emailInvalid'].message }}
        </p>
    </div>
    </div>
    <button type="submit" class="btn btn-primary"
(click)="submitted=true">Submit</button>
  </form>
<br/>
  <div [hidden]="!submitted">
    <h3> Employee Details </h3>
    <p>First Name: {{ registerForm.get('firstName')?.value }} </p>
    <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
    <p> Street: {{ registerForm.get('address.street')?.value }}</p>
    <p> Zip: {{ registerForm.get('address.zip')?.value }} </p>
    <p> City: {{ registerForm.get('address.city')?.value }}</p>
    <p>Email: {{ registerForm.get('email')?.value }}</p>
  </div>
</div>
```

2. Save the files and check the output in the browser

**Output:**



Registration Form

First Name

Alex

Last Name

Paul

Address

Street

lsb road

Zip

500032

City

Hyderabad

Email

alex.paul@abc.com

Employee Details

First Name: Alex

Last Name: Paul

Street: lsb road

Zip: 500032

## Exercise-8(a)

**Course Name:** Angular JS

**Module Name:** Custom Validators in Template Driven forms

Create a custom validator for the email field in the course registration form.

1. Write the code given below in **course.ts**

```
export class Course {
constructor(
public courseId: number,
public courseName: string,
public duration: string,
public email: string
) { }
}
```

2. In the **course-form.component.ts** file, pass a default value to the email field as shown below

```
import { Component } from '@angular/core';
import { Course } from './course';

@Component({
  selector: 'app-course-form',
  templateUrl: './course-form.component.html',
  styleUrls: ['./course-form.component.css']
})
export class CourseFormComponent {

  course: Course = new Course(1, 'Angular 2', '4 days', 'james@gmail.com');
  submitted = false;

  onSubmit() { this.submitted = true; }

}
```

3. Create a file with the name **email.validator.ts** under the course-form folder to implement custom validation functionality for the email field.

```
import { Directive } from '@angular/core';
import { NG_VALIDATORS, FormControl, Validator } from '@angular/forms';

@Directive({
  selector: '[validateEmail]',
  providers: [
    { provide: NG_VALIDATORS, useExisting: EmailValidator, multi: true },
  ],
})
export class EmailValidator implements Validator {
  validate(control: FormControl): any {
    const emailRegexp =
      /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/;
    if (!emailRegexp.test(control.value)) {
      return { emailInvalid: 'Email is invalid' };
    }
    return null;
  }
}
```

4. Add EmailValidator class in the root module i.e., **app.module.ts** as shown below

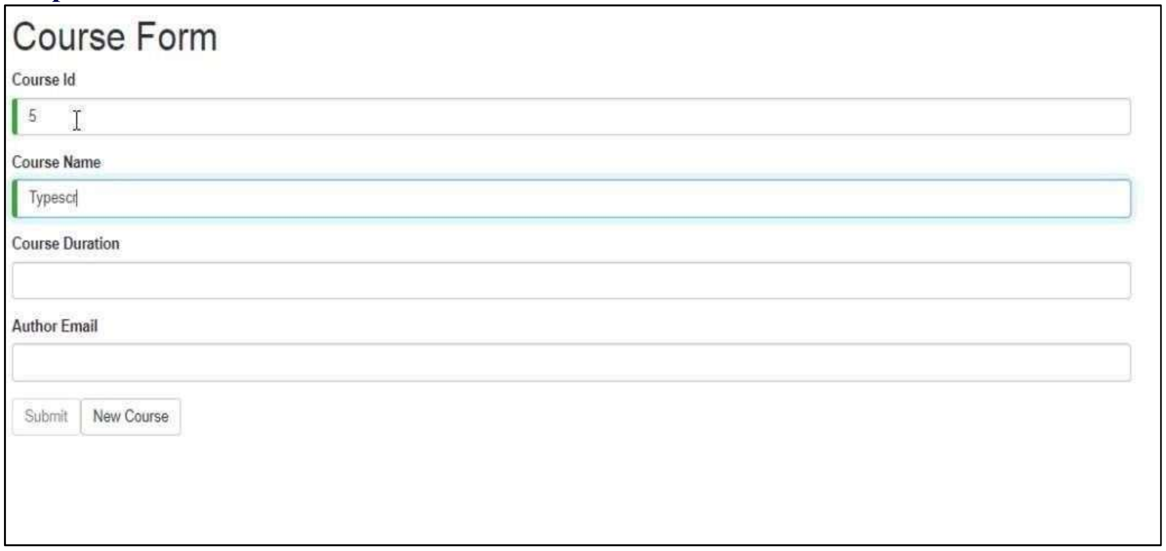
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { CourseFormComponent } from './course-form/course-form.component';
import { EmailValidator } from './course-form/email.validator';

@NgModule({
  declarations: [
  AppComponent,
  CourseFormComponent,
  EmailValidator
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

5. Add the following code in the **course-form.component.html** file for the email field as shown below

```
<div class="container">

  <div [hidden]="submitted">
    <h1>Course Form</h1>

    <form (ngSubmit)="onSubmit()"  #courseForm="ngForm">

      <div class="form-group">
        <label for="id">Course Id</label>
        <input type="text" class="form-control" required [(ngModel)]="course.courseId"
name="id" #id="ngModel">
        <div [hidden]="id.valid || id.pristine" class="alert alert-danger">
          Course Id is required</div>
      </div>
      <div class="form-group">
        <label for="name">Course Name</label>
        <input type="text" class="form-control" required [(ngModel)]="course.courseName"
          minlength="4" name="name" #name="ngModel">
        <div *ngIf="name.errors && (name.dirty || name.touched)" class="alert alert-
danger">
          <div [hidden]="!name.errors.required">Name is required</div>
          <div [hidden]="!name.errors.minlength">Name must be at least 4 characters
long.</div>
        </div>
      </div>

      <div class="form-group">
        <label for="duration">Course Duration</label>
        <input type="text" class="form-control" required [(ngModel)]="course.duration"
          name="duration" #duration="ngModel">
        <div   [hidden]="duration.valid   ||   duration.pristine"   class="alert   alert-
danger">Duration is required</div>
      </div>

      <div class="form-group">
        <label for="email">Author Email</label>
        <input type="text" class="form-control" required [(ngModel)]="course.email"
          name="email" #email="ngModel" validateEmail>

        <div *ngIf="email.errors && (email.dirty || email.touched)" class="alert alert-
danger">
          <div [hidden]="!email.errors.required">Email is required</div>
          <div [hidden]="!email.errors.emailInvalid">{{email.errors.emailInvalid}}</div>
        </div>
      </div>

      <button          type="submit"          class="btn          btn-primary"
[disabled]="!courseForm.form.valid">Submit</button>
      <button          type="button"          class="btn          btn-link"
(click)="courseForm.reset()">Reset</button>
    </form>
  </div>

  <div [hidden]="!submitted">
    <h2>You submitted the following:</h2>
    <div class="row">
      <div class="col-3">Course ID</div>
      <div class="col-9 pull-left">{{ course.courseId }}</div>
    </div>
    <div class="row">
      <div class="col-3">Course Name</div>
      <div class="col-9 pull-left">{{ course.courseName }}</div>
    </div>
    <div class="row">
      <div class="col-3">Duration</div>
      <div class="col-9 pull-left">{{ course.duration }}</div>
    </div>
    <div class="row">
      <div class="col-3">Email</div>
      <div class="col-9 pull-left">{{ course.email }}</div>
    </div>
    <br>
    <button class="btn btn-primary" (click)="submitted=false">Edit</button>
  </div>

</div>
```

6. Save the files and check the output in the browser

**Output:**



Course Form

Course Id

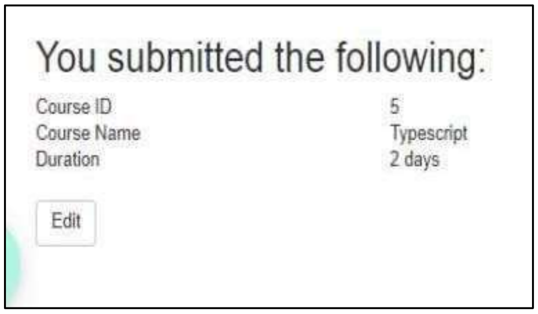
5

Course Name

Typescr

Course Duration

Author Email



You submitted the following:

| | |
|---|---|
| Course ID | 5 |
| Course Name | Typescript |
| Duration | 2 days |

## Exercise-8(b)

**Module Name:** Services Basics

Create a Book Component which fetches book details like id, name and displays them on the page in a list format. Store the book details in an array and fetch the data using a custom service.

1. Create **BookComponent** by using the following CLI command

```
D:/MyApp> ng generate component book
```

3. Create a file with the name **book.ts** under the book folder and add the following code.

```
export class Book {
   id!: number;
   name!: string;

         }
```

3. Create a file with the name **books-data.ts** under the book folder and add the following code.

```
import { Book } from './book';

export let BOOKS: Book[] = [
   { id: 1, name: 'HTML 5' },
   { id: 2, name: 'CSS 3' },
   { id: 3, name: 'Java Script' },
   { id: 4, name: 'Ajax Programming' },
   { id: 5, name: 'jQuery' },
   { id: 6, name: 'Mastering Node.js' },
   { id: 7, name: 'Angular JS 1.x' },
   { id: 8, name: 'ng-book 2' },
   { id: 9, name: 'Backbone JS' },
   { id: 10, name: 'Yeoman' }
];
```

4. Create a service called **BookService** under the book folder using the following CLI command

```
D:\MyApp\src\app\book>ng generate service book
```

5. Add the following code in **book.service.ts**

```
import { Injectable } from '@angular/core';
import { BOOKS } from './books-data';
   @Injectable({
 providedIn: 'root'
})
   export class BookService {
 getBooks() {
  return BOOKS;
}}
```

6. Add the following code in the **book.component.ts** file

```
import { Component, OnInit } from '@angular/core';
import { Book } from './book';
import { BookService } from './book.service';
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
  books!: Book[];
  constructor(private bookService: BookService) { }
  getBooks() {
    this.books = this.bookService.getBooks();
  }
  ngOnInit() {
    this.getBooks();
  }}
```

7. Write the below-given code in **book.component.html**

```
<h2>My Books</h2>
<ul class="books">
  <li *ngFor="let book of books">
    <span class="badge">{{book.id}}</span> {{book.name}}

  </li></ul>
```

8. Add the following code in **book.component.css** which has styles for books

```
.books {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 13em;
}
.books li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #eee;
  margin: 0.5em;
  padding: 0.3em 0;
  height: 1.5em;
  border-radius: 4px;
}
.books li:hover {
  color: #607d8b;
  background-color: #ddd;
  left: 0.1em;
}
.books .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607d8b;
  line-height: 0.5em;
  position: relative;
```

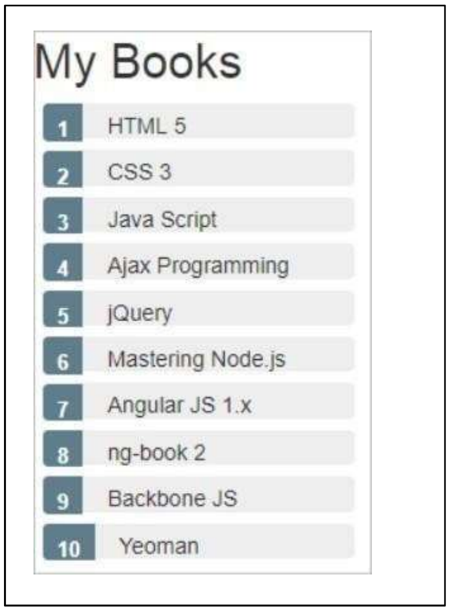
```
left: -1px;

top: -4px;

height: 1.8em;

margin-right: 0.8em;

border-radius: 4px 0 0 4px;


        }
```

9. Add the following code in app.component.html

   `<app-book></app-book>`

10. Save the files and check the output in the browser

**Output:**



My Books

1 HTML 5
2 CSS 3
3 Java Script
4 Ajax Programming
5 jQuery
6 Mastering Node.js
7 Angular JS 1.x
8 ng-book 2
9 Backbone JS
10 Yeoman

## Exercise-8(c)

**Module Name:** RxJS Observables

Create and use an observable in Angular.

**app.component.ts**

```
import { Component } from '@angular/core';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-root',

  styleUrls: ['./app.component.css'],

  templateUrl: './app.component.html'

})

export class AppComponent {

  data!: Observable<number>;

  myArray: number[] = [];

  errors!: boolean;

  finished!: boolean;

  fetchData(): void {

    this.data = new Observable(observer => {

      setTimeout(() => { observer.next(11); }, 1000),

      setTimeout(() => { observer.next(22); }, 2000),

      setTimeout(() => { observer.complete(); }, 3000);

    });this.data.subscribe((value) => this.myArray.push(value),

      error => this.errors = true,

      () => this.finished = true);}
```

Line 2: imports Observable class from rxjs module

Line 11: data is of type Observable which holds numeric values

Line 16: fetchData() is invoked on click of a button

Line 17: A new Observable is created and stored in the variable data

Line 18-20: next() method of Observable sends the given data through the stream. With a delay of 1,2 and 3 seconds, a stream of numeric values will be sent. Complete() method completes the Observable stream i.e., closes the stream.

Line 22: Observable has another method called subscribe which listens to the data coming through the stream. Subscribe() method has three parameters. The first parameter is a success callback which will be invoked upon receiving successful data from the stream. The second parameter is an error callback which will be invoked when Observable returns an error and the third parameter is a complete callback which will be invoked upon successful streaming of values from Observable i.e., once complete() is invoked. After which the successful response, the data is pushed to the local array called myArray, if any error occurs, a Boolean value called true is stored in the errors variable and upon complete() will assign a Boolean value true in a finished variable.

**app.component.html**

```
<b> Using Observables!</b>

<h6 style="margin-bottom: 0">VALUES:</h6>

<div *ngFor="let value of myArray">{{ value }}</div>

<div style="margin-bottom: 0">ERRORS: {{ errors }}</div>

<div style="margin-bottom: 0">FINISHED: {{ finished }}</div>


<button style="margin-top: 2rem" (click)="fetchData()">Fetch Data</button>
```

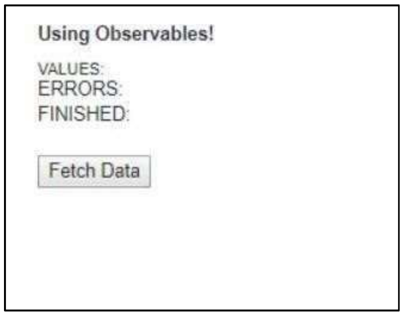
Line 4: ngFor loop is iterated on myArray which will display the values on the page

Line 6: {{ errors }} will render the value of errors property if any

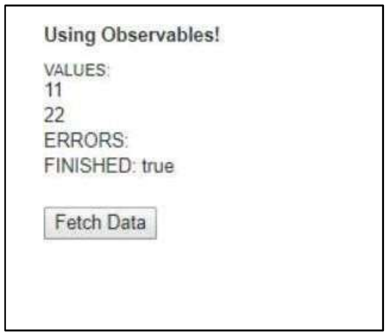
Line 8: Displays finished property value when complete() method of Observable is executed

Line 10: Button click event is bound with fetchData() method which is invoked and creates an observable with a stream of numeric values

**Output:**



```
Using Observables!
VALUES:
ERRORS:
FINISHED:
```



```
Using Observables!
VALUES:
11
22
ERRORS:
FINISHED: true
```

## EXERCISE-9(a)

**Module Name:** Server Communication using HttpClient

Create an application for Server Communication using HttpClient

- In the example used for custom services concept, add HttpModule to the **app.module.ts** to make use of HttpClient class.

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import {HttpClientModule} from '@angular/common/http';

import { AppComponent } from './app.component';

import { BookComponent } from './book/book.component';

@NgModule({

  imports: [BrowserModule, HttpClientModule],

  declarations: [AppComponent, BookComponent],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

- Add the following code in **book.service.ts** file

```
import { Injectable } from '@angular/core';

import { HttpClient, HttpErrorResponse, HttpHeaders } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError, tap } from 'rxjs/operators';

import { Book } from './book';

@Injectable({

    providedIn:'root'

})
```

```
export class BookService {
  booksUrl = 'http://localhost:3020/bookList';
  constructor(private http: HttpClient) { }
  getBooks(): Observable<Book[]> {
    return this.http.get<Book[]>('http://localhost:3020/bookList').pipe(
      tap((data: any) => console.log('Data Fetched:' + JSON.stringify(data))),
      catchError(this.handleError));
  }
  addBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type': 'application/json' });
    return this.http.post('http://localhost:3020/addBook', book, { headers: options }).pipe(
      catchError(this.handleError));
  }
  updateBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type': 'application/json' });
    return this.http.put<any>('http://localhost:3020/update', book, { headers: options }).pipe(
      tap((_: any) => console.log(`updated hero id=${book.id}`)),
      catchError(this.handleError)
    );
  }
  deleteBook(bookId: number): Observable<any> {
    const url = `${this.booksUrl}/${bookId}`;
    return this.http.delete(url).pipe(
    catchError(this.handleError));
  }
  private handleError(err: HttpErrorResponse): Observable<any> {
    let errMsg = '';
    if (err.error instanceof Error) {
      // A client-side or network error occurred. Handle it accordingly.
      console.log('An error occurred:', err.error.message);
      errMsg = err.error.message;
    } else {
      // The backend returned an unsuccessful response code.
      // The response body may contain clues as to what went wrong,
      console.log(`Backend returned code${err.status}`);
      errMsg = err.error.status;
    }
    return throwError(()=>errMsg);
  }
}
```

- Write the code given below in **book.component.ts**

```
import { Component, OnInit } from '@angular/core';
import { BookService } from './book.service';
import { Book } from './book';
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
  title = 'Demo on HttpClientModule';
  books!: Book[];
```

```
errorMessage!: string;
ADD_BOOK!: boolean;
UPDATE_BOOK!: boolean;
DELETE_BOOK!: boolean;
constructor(private bookService: BookService) { }
getBooks() {
this.bookService.getBooks().subscribe({
   next:  books => this.books = books,
   error:error => this.errorMessage = <any>error
  })
}
addBook(bookId: string, name: string): void {
  let id=parseInt(bookId)
  this.bookService.addBook({id, name })
   .subscribe({next:(book: any) => this.books.push(book)});
}
updateBook(bookId: string, name: string): void {
  let id=parseInt(bookId)
  this.bookService.updateBook({ id, name })
   .subscribe({next:(book: any) => this.books = book});
}
deleteBook(bookId: string): void {
  let id=parseInt(bookId)
  this.bookService.deleteBook(id)
   .subscribe({next:(book: any) => this.books = book});
}
```

```
  ngOnInit() {
    this.getBooks();
  }
}
```

- Write the code given below in **book.component.html**

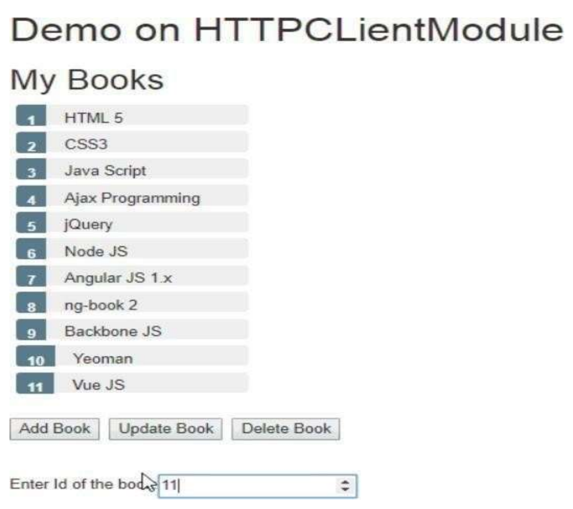
```
<h2>{{ title }}</h2>
<h2>My Books</h2>

<ul class="books">
  <li *ngFor="let book of books">
    <span class="badge">{{ book.id }}</span> {{ book.name }}
  </li>
</ul>
<button class="btn btn-primary" (click)="ADD_BOOK = true">Add Book</button>&nbsp;
<button class="btn btn-primary" (click)="UPDATE_BOOK = true">Update Book</button>&nbsp;
<button class="btn btn-primary" (click)="DELETE_BOOK = true">Delete Book</button>
<br />
<div *ngIf="ADD_BOOK">
  <table>
    <tr>
      <td>Enter Id of the book:</td>
      <td>
        <input type="number" #id />
      </td>
    </tr>
```

```
      <br />
      <tr>
        <td>Enter Name of the Book:</td>
        <td>
          <input type="text" #name />
          <br />
        </td>
      </tr>
      <br />
      <tr>
        <td>
          <button class="btn btn-primary" (click)="addBook(id.value, name.value); ADD_BOOK =
false">
            Add Record
          </button>
        </td>
      </tr>
    </table>

    <br />
  </div>
  <div *ngIf="UPDATE_BOOK">
    <table>
      <tr>
        <td>Enter Id of the book:</td>
        <td>
          <input type="number" #id />
        </td>
      </tr>
      <br />
      <tr>
        <td>Enter Name of the Book:</td>
        <td>
          <input type="text" #name />
          <br />
        </td>
      </tr>
      <br />

      <tr>
        <td>
          <button class="btn btn-primary" (click)="updateBook(id.value, name.value);
  UPDATE_BOOK = false">
            Update Record
          </button>
        </td>
      </tr>
    </table>
  </div>
  <br />

  <div *ngIf="DELETE_BOOK">
```

```
  <table>
    <tr>
      <td>Enter Id of the book:</td>
      <td>
        <input type="number" #id />
      </td>
    </tr>
    <br />

    <tr>
      <td>
        <button class="btn btn-primary" (click)="deleteBook(id.value); DELETE_BOOK = false">
          Delete Record
        </button>
      </td>
    </tr>
  </table>
</div>

<div class="error" *ngIf="errorMessage">{{ errorMessage }}</div>
```

- Save the files and check the output in the browser

**Output:**



Demo on HTTPCLientModule

My Books

1 HTML 5
2 CSS3
3 Java Script
4 Ajax Programming
5 jQuery
6 Node JS
7 Angular JS 1.x
8 ng-book 2
9 Backbone JS
10 Yeoman
11 Vue JS

Enter Id of the book: 11

## Excícisc-9(b)

**Module Name**: Communicating with different backend services using Angular HttpClient Create a custom service called ProductService in which Http class is used to fetch data stored in the JSON files

Product-list.component.ts:

import { Component, OnInit } from '@angular/core'; import { ProductService } from './product.service'; @Component({

selector: 'app-product-list',

templateUrl: './product-list.component.html', styleUrls: ['./product-list.component.css']

})

export class ProductListComponent implements OnInit { products: any[] = [];

constructor(private productService: ProductService) { } ngOnInit() { this.productService.getProducts().subscribe((data) => {

this.products = data;

}); } }

**Product.service.ts:**

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http'; import { Observable } from 'rxjs';

@Injectable({ providedIn: 'root' })

export class ProductService {

private productsUrl = 'assets/products.json'; constructor(private http: HttpClient) { }

getProducts(): Observable<any[]> {

return this.http.get<any[]>(this.productsUrl); } }

**App.component.html:**

<div *ngIf="products.length > 0; else noProducts">

<h2>Products</h2> <ul>

<li *ngFor="let product of products">

```
<h3>{{ product.name }}</h3>

<p>Price: {{ product.price }}</p>

<p>Description: {{ product.description }}</p>

</li> </ul> </div>

<ng-template #noProducts>

<p>No products available</p>

</ng-template>
```

**App.component.ts:**

```
import { Component, OnInit } from '@angular/core';

import { ProductService } from './product-list/product.service';

@Component({ selector: 'my-app',

templateUrl: './app.component.html', styleUrls: ['./app.component.css'] })

export class AppComponent implements OnInit { products: any[] = [];

constructor(private productService: ProductService) { } ngOnInit() {

this.productService.getProducts().subscribe((data) => { this.products = data;}); } }
```

**App.module.ts:**

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser'; import { HttpClientModule } from
'@angular/common/http';

import { AppComponent } from './app.component'; @NgModule({

declarations: [ AppComponent ],

imports: [ BrowserModule, HttpClientModul ],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }
```

**Exercise-10(a)**

**Module Name:** Routing Basics, Router Links

Create multiple components and add routing to provide navigation between them

1. Consider the example used for the **HttpClient** concept.
2. Create another component with the name dashboard using the following command

```
D:\MyApp>ng generate component dashboard
```

3. Open **dashboard.component.ts** and add the following code

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  books: Book[] = [];
  constructor(
    private router: Router,
    private bookService: BookService) { }
  ngOnInit(): void {
  this.bookService.getBooks()
      .subscribe({next:books => this.books = books.slice(1, 5)});
  }gotoDetail(book: Book): void {this.router.navigate(['/detail', book.id]);}}
```

4. Open **dashboard.component.html** and add the following code

```
<h3>Top Books</h3>
<div class="grid grid-pad">
  <div *ngFor="let book of books" (click)="gotoDetail(book)" class="col-1-4">
    <div class="module book">
      <h4>{{ book.name }}</h4>
    </div>
  </div>
</div>
```

5. Open **dashboard.component.css** and add the following code

```
[class*="col-"] {
  float: left;
}
*,
*:after,
*:before {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
h3 {
  text-align: center;
  margin-bottom: 0;
}
[class*="col-"] {
  padding-right: 20px;
  padding-bottom: 20px;
}
[class*="col-"]:last-of-type {
  padding-right: 0;
}
.grid {
  margin: 0;
}
.col-1-4 {
  width: 25%;
}
.module {
  padding: 20px;
  text-align: center;
  color: #eee;
  max-height: 120px;
  min-width: 120px;
  background-color: #607d8b;
  border-radius: 2px;
}
h4 {
  position: relative;
}
.module:hover {
  background-color: #eee;
  cursor: pointer;
  color: #607d8b;
}
.grid-pad {
  padding: 10px 0;
}
.grid-pad > [class*="col-"]:last-of-type {
  padding-right: 20px;
}
@media (max-width: 600px) {
  .module {
    font-size: 10px;
    max-height: 75px;
  }
}
@media (max-width: 1024px) {
  .grid {
    margin: 0;
  }
  .module {
    min-width: 60px;
  }
}
```

6. Create another component called **book-detail** using the following command

```
D:\MyApp>ng generate component bookDetail
```

7. Open **book.service.ts** and add **getbook()** method as shown below to fetch specific book details

```
import { Injectable } from '@angular/core';

import { HttpClient, HttpErrorResponse, HttpHeaders, HttpResponse } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError, tap, map} from 'rxjs/operators';

import { Book } from './book';

@Injectable({

  providedIn:'root'

})

export class BookService {

 booksUrl = 'http://localhost:3020/bookList';

 private txtUrl = './assets/sample.txt';

 constructor(private http: HttpClient) { }

 getBooks(): Observable<Book[]> {

  return this.http.get<any>(this.booksUrl, {observe:'response'}).pipe(

   tap((data: any) => console.log('Data Fetched:' + JSON.stringify(data))),

   catchError(this.handleError));

 }

 getBook(id: any) {

  return this.getBooks().pipe(

    map((books) => books.find((book) => book.id == id))

  );

 }
```

```
addBook(book: Book): Observable<any> {
  const options = new HttpHeaders({ 'Content-Type': 'application/json' });
  return this.http.post('http://localhost:3020/addBook', book, { headers: options }).pipe(
    catchError(this.handleError));
}
updateBook(book: Book): Observable<any> {
  const options = new HttpHeaders({ 'Content-Type': 'application/json' });
  return this.http.put<any>('http://localhost:3020/update', book, { headers: options }).pipe(
    tap((_: any) => console.log(`updated hero id=${book.id}`)),
    catchError(this.handleError)
  );
}
deleteBook(bookId: number): Observable<any> {
  const url = `${this.booksUrl}/${bookId}`;
  return this.http.delete(url).pipe(
  catchError(this.handleError));
}
private handleError(err: HttpErrorResponse): Observable<any> {
  let errMsg = '';
  if (err.error instanceof Error) {
    // A client-side or network error occurred. Handle it accordingly.
    console.log('An error occurred:', err.error.message);
    errMsg = err.error.message;
  } else {
    // The backend returned an unsuccessful response code.
    // The response body may contain clues as to what went wrong,
      console.log(`Backend returned code${err.status}`);
      errMsg = err.error.status;
    }
    return throwError(()=>errMsg);
  }
}
```

7. Open **book-detail.component.ts** and add the following code

```
        import { Component, OnInit } from '@angular/core';
        import { ActivatedRoute } from '@angular/router';
        import { Book } from '../book/book';
        import { BookService } from '../book/book.service';
        @Component({
        selector: 'app-book-detail',
templateUrl: './book-detail.component.html',
styleUrls: ['./book-detail.component.css'],
})
export class BookDetailComponent implements OnInit {
book!: Book;
error!: any;
constructor(
private bookService: BookService,
private route: ActivatedRoute
) { }
ngOnInit() {
this.route.paramsMap.subscribe(params => {
this.bookService.getBook(params.get('id')).subscribe((book)  =>  {
```

```
this.book = book ?? this.book;
});
});
}
goBack() {
window.history.back();

}}
```

9. Open **book-detail.component.html** and add the following code

```
<div *ngIf="book">
  <h2>{{ book.name }} details!</h2>
  <div><label>id: </label>{{ book.id }}</div>
  <div>
    <label>name: </label> <input [(ngModel)]="book.name" placeholder="name" />
  </div>
  <button (click)="goBack()">Back</button>
</div>
```

10. Open **book-detail.component.css** and add the following code

```
label {
  display: inline-block;
  width: 3em;
  margin: 0.5em 0;
  color: #607d8b;
  font-weight: bold;
}
```

```
input {
  height: 2em;
  font-size: 1em;
  padding-left: 0.4em;
}
button {
  margin-top: 20px;
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #ccc;
  cursor: auto;
}
```

11. Generate PageNotFound component using the following CLI command

```
D:\MyApp>ng g c PageNotFound
```

12. Add below code to page-not-found.component.html:

```
<div>
   <h1>404 Error</h1>
   <h1>Page Not Found</h1>
</div>
```

13. Add the below code to app-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
    const appRoutes: Routes = [
   { path: 'dashboard', component: DashboardComponent },
   { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
   { path: 'books', component: BookComponent },
   { path: 'detail/:id', component: BookDetailComponent },
   { path: '**', component: PageNotFoundComponent },
];
@NgModule({
   imports: [
      RouterModule.forRoot(appRoutes)
   ],
   exports: [
      RouterModule
   ]
})
```

```
export class AppRoutingModule { }
```

14. Write the below-given code in **app.module.ts**

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpClientModule } from '@angular/common/http';

import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

import { BookComponent } from './book/book.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { BookDetailComponent } from './book-detail/book-detail.component';

import { AppRoutingModule } from './app-routing.module';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';


@NgModule({

  imports: [BrowserModule, HttpClientModule, FormsModule, AppRoutingModule],

  declarations: [AppComponent, BookComponent, DashboardComponent, BookDetailComponent,
PageNotFoundComponent],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

15. Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'Tour of Books';
}
```

16. Write the below-given code in **app.component.html**

```
<h1>{{title}}</h1>
<nav>
   <a [routerLink]='["/dashboard"]' routerLinkActive="active">Dashboard</a>
   <a [routerLink]='["/books"]' routerLinkActive="active">Books</a>
</nav>
<router-outlet></router-outlet>
```

17. Open **app.component.css** and add the following code

```
         h1 {
         color: #369;
         font-family: Arial, Helvetica, sans-serif;
         font-size: 250%;
         }
         h2, h3 {
    color: #444;
    font-family: Arial, Helvetica, sans-serif;
    font-weight: lighter;
    }
    body {
```

```
margin: 2em;

}

body, input[text], button {

color: #888;

font-family: Cambria, Georgia;

}

a {

cursor: pointer;

cursor: hand;

}

button {

font-family: Arial;

background-color: #eee;

border: none;

padding: 5px 10px;

border-radius: 4px;

cursor: pointer;

cursor: hand;

}

button:hover {

background-color: #cfd8dc;

}

button:disabled {

background-color: #eee;

color: #aaa;

cursor: auto;
```

```
}
39.
/*Navigation link styles */
nav a {
padding: 5px 10px;
text-decoration: none;
margin-right: 10px;
margin-top: 10px;
display: inline-block;
background-color: #eee;
border-radius: 4px;
}
nav a:visited, a:link {
color: #607D8B;
}
nav a:hover {
color: #039be5;
background-color: #CFD8DC;
}
nav a.active {
color: #039be5;
}
* {
font-family: Arial, Helvetica, sans-serif;
}
```

18. Open **styles.css** under the src folder and add the following code

```
body{
  padding:10px;
}
```

19. Open **book.component.ts** file in book folder and add the following code

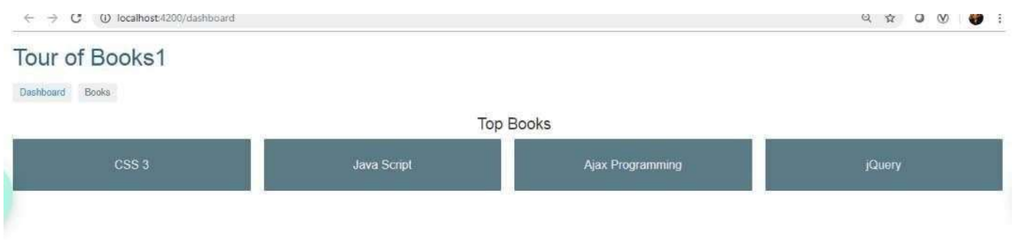
```
import { Component, OnInit } from '@angular/core';
import { Book } from './book';
import { BookService } from './book.service';
@Component({
selector: 'app-book',
templateUrl: './book.component.html',
styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
10.
books!: Book[];
errorMessage!: string;
constructor(private bookService: BookService) { }
getBooks() {
this.bookService.getBooks().subscribe({
next:  books => this.books = books,
error:error => this.errorMessage = <any>error
})
}
ngOnInit(): void{
this.getBooks()}}
```

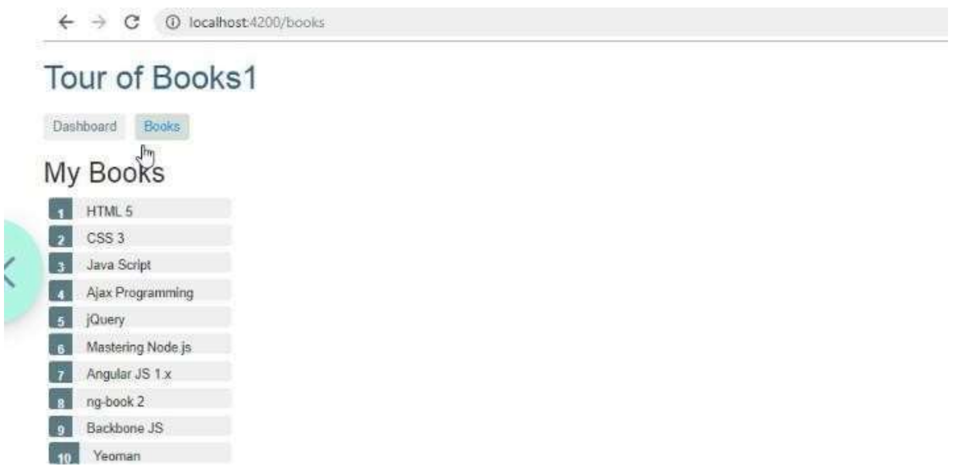
20. Open **book.component.html** and update with below code.

```
<h2>My Books</h2>

<ul class="books">

  <li *ngFor="let book of books">

    <span class="badge">{{ book.id }}</span> {{ book.name }}

  </li>

</ul>

<div class="error" *ngIf="errorMessage">{{ errorMessage }}</div>
```

Save the files and check the output in the browser.

**Output:**





**Exercise-10(b)**

**Module Name:** Route Guards

Considering the same example used for routing, add route guard to BooksComponent. Only after logging in, the user should be able to access BooksComponent. If the user tries to give the URL of Bookscomponent in another tab or window, or if the user tries

1. Create a component named **LoginComponent** and add the following code to the **login.component.html** file:

```
<h3 style="position: relative; left: 60px">Login Form</h3>

<div *ngIf="invalidCredentialMsg" style="color: red">

  {{ invalidCredentialMsg }}

</div>

<br />

<div style="position: relative; left: 20px">

  <form[formGroup]="loginForm" (ngSubmit)="onFormSubmit()">

    <p>User Name <input formControlName="username" /></p>

    <p>

      Password

      <input

        type="password"

        formControlName="password"

        style="position: relative; left: 10px"

      />

    </p>

    <p><button type="submit">Submit</button></p>

  </form>

</div>
```

2. Add the following code to the **login.component.ts** file:

```
import { Component } from '@angular/core';

import { FormBuilder, FormGroup } from'@angular/forms';

import { Router } from '@angular/router';

import { LoginService } from './login.service';

@Component({

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.css'],

})

export class LoginComponent {

  invalidCredentialMsg!: string;

  loginForm!: FormGroup;

  constructor(

    private loginService: LoginService,

    private router: Router,

    private formbuilder: FormBuilder

  ) {

    this.loginForm = this.formbuilder.group({

      username: [],

      password: [],

    });

  }

  onFormSubmit(): void {

    const uname = this.loginForm.value.username;

    const pwd = this.loginForm.value.password;

    this.loginService
```

```
      .isUserAuthenticated(uname,  pwd)
      .subscribe({next:(authenticated) => {
        if (authenticated) {
        this.router.navigate(['/books']);
        } else {
          this.invalidCredentialMsg = 'Invalid Credentials.  Try again.';
        }
      }});
  }
}
```

3. Create user.ts file under login folder and add the following code to **user.ts** file:

```
export class User {
  constructor(public userId: number, public username: string, public password: string) { }
}
```

4. Add the following code to the **login.service.ts** file inside login folder:

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { map } from 'rxjs/operators';
import { User } from './user';
const USERS = [
    new User(1, 'user1', 'user1'),
    new User(2, 'user2', 'user2')
];
const usersObservable = of(USERS);
@Injectable({
providedIn: 'root'
})

export class LoginService {

  private isloggedIn = false;

  getAllUsers(): Observable<User[]> {

    return usersObservable;

  }

  isUserAuthenticated(username: string, password: string): Observable<boolean> {

    return this.getAllUsers().pipe(

      map(users => {

        const Authenticateduser = users.find(user => (user.username === username) &&
(user.password === password));

        if (Authenticateduser) {

          this.isloggedIn = true;

        } else {

          this.isloggedIn = false;

        }

        return this.isloggedIn;

      })

    );

  }

  isUserLoggedIn(): boolean {

    return this.isloggedIn;

  }

}
```

5. Create another service class called **login-guard.service** inside login folder and add the following code:

```
import { Injectable } from '@angular/core';

import { CanActivate, Router } from '@angular/router';

import { LoginService } from './login.service';

@Injectable({

  providedIn: 'root'

})

export class LoginGuardService implements CanActivate {

  constructor(private loginService: LoginService, private router: Router) { }

  canActivate(): boolean {

    if (this.loginService.isUserLoggedIn()) {

      return true;}

    this.router.navigate(['/login']);

    return false;}
```

6. Add the following code in **app.module.ts:**

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpClientModule } from '@angular/common/http';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

import { BookComponent } from './book/book.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { BookDetailComponent } from './book-detail/book-detail.component';

import { AppRoutingModule } from './app-routing.module';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

import { LoginComponent } from './login/login.component';
```

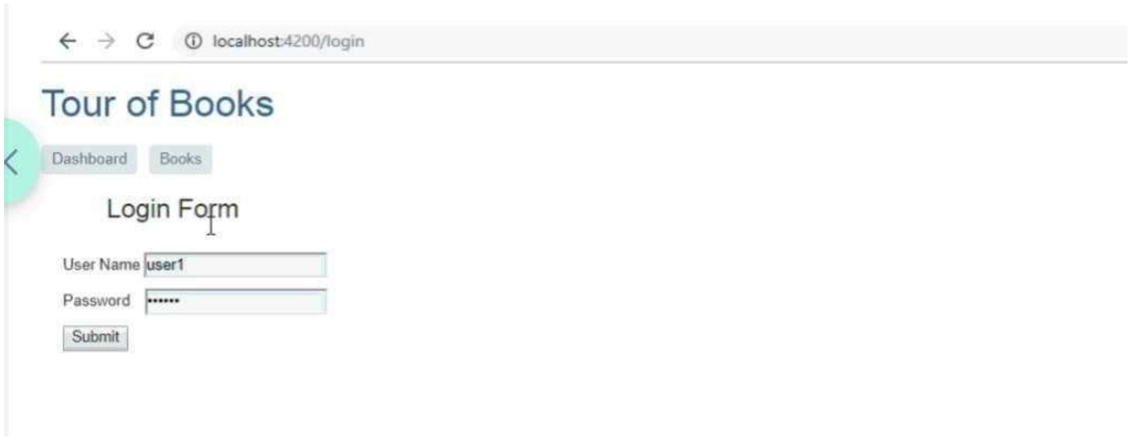
```
@NgModule({

 imports: [BrowserModule, HttpClientModule, ReactiveFormsModule,FormsModule,
AppRoutingModule],

 declarations: [AppComponent, LoginComponent, BookComponent, DashboardComponent,
BookDetailComponent, PageNotFoundComponent],

  providers: [],

 bootstrap: [AppComponent]})

export class AppModule { }
```

7. Add the following code to **app-routing.module.ts:**

```
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { BookComponent } from './book/book.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { BookDetailComponent } from './book-detail/book-detail.component';

import { PageNotFoundComponent } from'./page-not-found/page-not-found.component';

import { LoginGuardService } from './login/login-guard.service';

import { LoginComponent } from './login/login.component';

const appRoutes: Routes = [

    { path: 'dashboard', component: DashboardComponent },

    { path: '', redirectTo: '/dashboard', pathMatch: 'full' },

    { path: 'books', component: BookComponent , canActivate:[LoginGuardService] },

    { path: 'detail/:id', component: BookDetailComponent },

    {path: 'login',component:LoginComponent},

    { path: '**', component: PageNotFoundComponent },];

@NgModule({

    imports: [

        RouterModule.forRoot(appRoutes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }
```

1. Save the files and check the output in the browser.

**Output:**


localhost:4200/login

Tour of Books

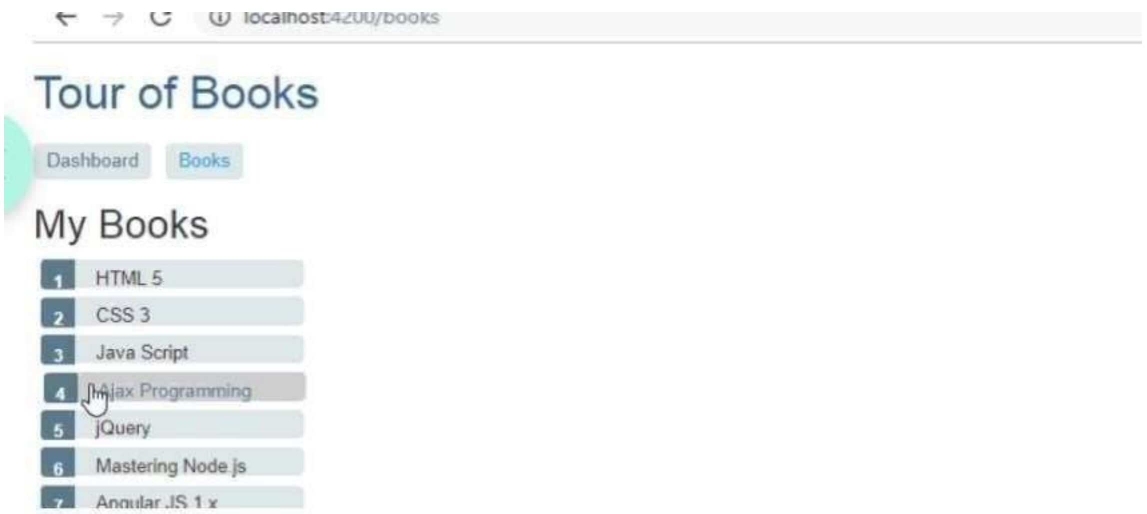
Dashboard Books

Login Form

User Name user1

Password ••••••

Submit

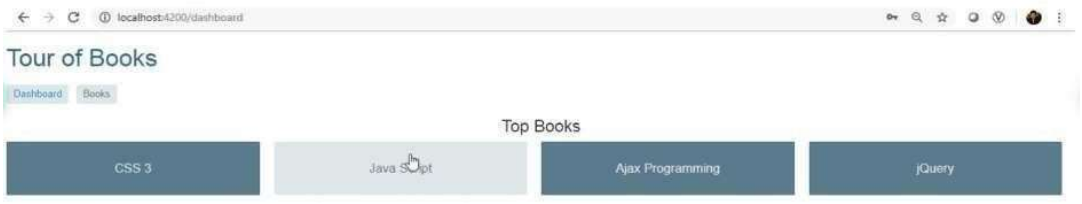

localhost:4200/books

Tour of Books

Dashboard Books

My Books

1 HTML 5
2 CSS 3
3 Java Script
4 Ajax Programming
5 jQuery
6 Mastering Node js
7 Angular JS 1.x


localhost:4200/dashboard

Tour of Books

Dashboard Books

Top Books

CSS 3 Java Script Ajax Programming jQuery

## Exercise-10(c)

**Module Name:** Asynchronous Routing

Apply lazy loading to BookComponent. If lazy loading is not added to the demo, it has loaded in 1.14 s. Observe the load time at the bottom of the browser console. Press F12 in the browser and click the Network tab and check the Load time.

1. Write the code given below in the **book-routing.module.ts** file inside book folder.

```
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { BookComponent } from './book.component';

import { LoginGuardService } from '../login/login-guard.service';

const bookRoutes: Routes = [

  {

    path: '',

    component: BookComponent,

    canActivate: [LoginGuardService]

  }

];

@NgModule({

  imports: [RouterModule.forChild(bookRoutes)],

  exports: [RouterModule]

})

export class BookRoutingModule { }
```

2. Create the **book.module.ts** file inside book folder and add the following code

```
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { BookComponent } from './book.component';

import { BookRoutingModule } from './book-routing.module';
```

```
@NgModule({
  imports: [CommonModule, BookRoutingModule],
  declarations: [BookComponent]
})
export class BookModule { }
```

3. Add the following code to the **app-routing.module.ts** file

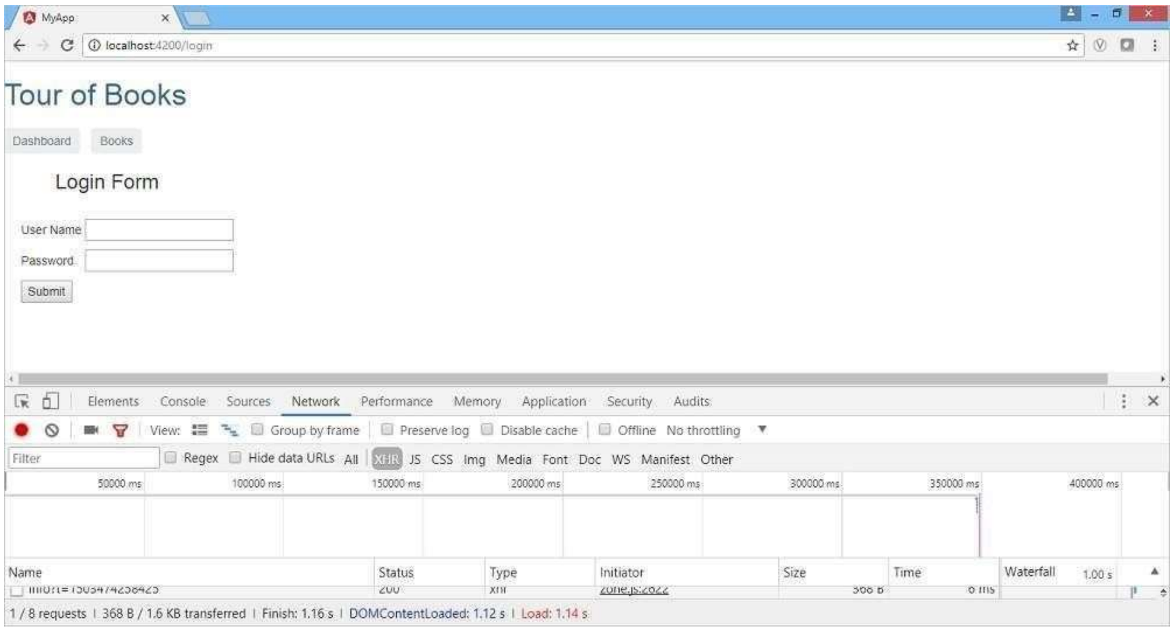
```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { LoginGuardService } from './login/login-guard.service';
import { LoginComponent } from './login/login.component';
import { PageNotFoundComponent } from'./page-not-found/page-not-found.component';
const appRoutes: Routes = [
    { path: '', redirectTo: '/login', pathMatch: 'full' },
    { path: 'login', component: LoginComponent },
    { path: 'books', loadChildren: () => import('./book/book.module').then(m => m.BookModule) },
    { path: 'dashboard', component: DashboardComponent },
    { path: 'detail/:id', component: BookDetailComponent} ,
    { path: '**', component: PageNotFoundComponent }
];
@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }
```

4. Add the following code to the **app.module.ts** file

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { AppRoutingModule } from './app-routing.module';
import { PageNotFoundComponent } from'./page-not-found/page-not-found.component';
import { LoginComponent } from './login/login.component';
@NgModule({
 imports: [BrowserModule, HttpClientModule, ReactiveFormsModule,FormsModule,
AppRoutingModule],
 declarations: [AppComponent, LoginComponent, DashboardComponent, BookDetailComponent,
PageNotFoundComponent],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

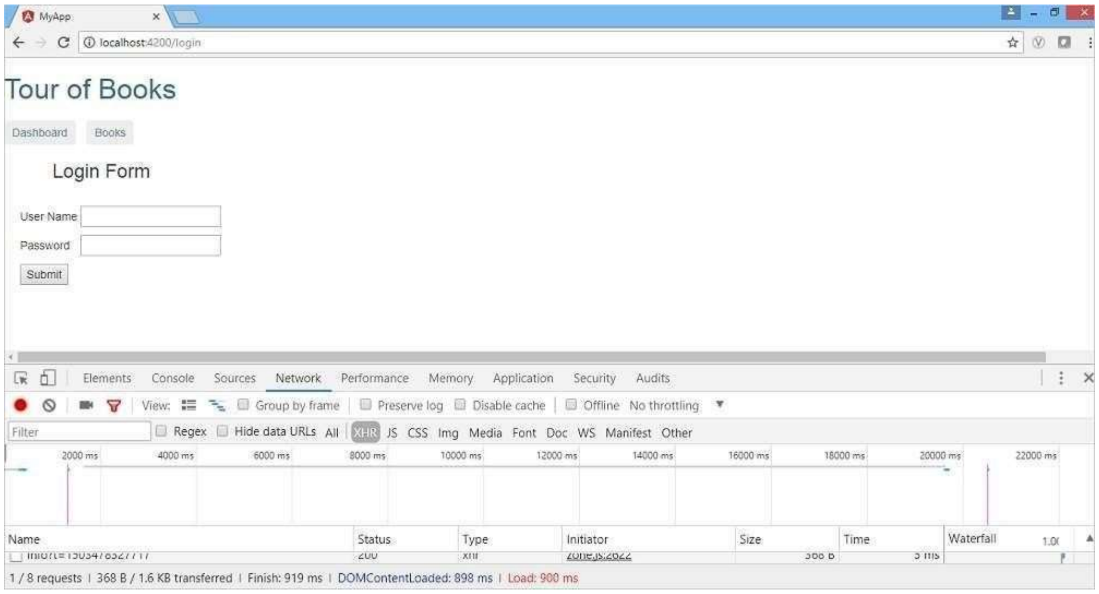
Save the files and check the output in the browser

**Output:**



1 / 8 requests | 368 B / 1.6 KB transferred | Finish: 1.16 s | DOMContentLoaded: 1.12 s | Load: 1.14 s

If lazy loading is added to the demo, it has loaded in 900 ms. As BookComponent will be loaded after login, the load time is reduced initially.



1 / 8 requests | 368 B / 1.6 KB transferred | Finish: 919 ms | DOMContentLoaded: 898 ms | Load: 900 ms

## Exercise-10(d)

**Module Name:** Nested Routes

Implement Child Routes to a submodule.

1. Write the following code in **app.module.ts**.

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpClientModule } from '@angular/common/http';

import { ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

import { AppRoutingModule } from './app-routing.module';

import { LoginComponent } from './login/login.component';

@NgModule({

  imports: [BrowserModule, HttpClientModule,  ReactiveFormsModule, AppRoutingModule],

  declarations: [AppComponent, LoginComponent],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

2. Write the following code in **app-routing.module.ts**.

```
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { LoginComponent } from './login/login.component';

import { PageNotFoundComponent } from'./page-not-found/page-not-found.component';

const appRoutes: Routes = [

    { path: '', redirectTo: '/login', pathMatch: 'full' },
    { path: 'login', component: LoginComponent },
    { path: 'books', loadChildren: () => import('./book/book.module').then(m => m.BookModule) },
    { path: '**', component: PageNotFoundComponent }
];
@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
    exports: [
        RouterModule
    ]
})
export class AppRoutingModule { }
```

3. Write the following code in **app.component.html**

```
<h1>{{title}}</h1>
<nav>
    <a [routerLink]='["/books"]' routerLinkActive="active">Books</a>
    <a [routerLink]='["/books/dashboard"]' routerLinkActive="active">Dashboard</a>
</nav>
<router-outlet></router-outlet>
```

4. Write the below code in **book.module.ts**

```
 import { NgModule } from '@angular/core';
import { BookComponent } from './book.component';
import { BookRoutingModule } from './book-routing.module';
import { FormsModule } from '@angular/forms';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { DashboardComponent } from '../dashboard/dashboard.component';
import { CommonModule } from '@angular/common';
@NgModule({
  imports: [ CommonModule, BookRoutingModule, FormsModule],
  declarations: [BookComponent, BookDetailComponent, DashboardComponent]
})
export class BookModule { }
```

5. Write the following code in **book-routing.module.ts**.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookComponent } from './book.component';
import { LoginGuardService } from '../login/login-guard.service';
import { DashboardComponent } from '../dashboard/dashboard.component';
import { BookDetailComponent } from '../book-detail/book-detail.component';
const bookRoutes: Routes = [
    {
      path: '',
      component: BookComponent,
      children: [
        { path: 'dashboard', component: DashboardComponent },
        { path: 'detail/:id', component: BookDetailComponent }
      ],
      canActivate: [LoginGuardService]
    }];
@NgModule({
    imports: [RouterModule.forChild(bookRoutes)],
  exports: [RouterModule]
})
export class BookRoutingModule { }
```

6. Write the below code in **book.component.html**

```
<br/>
    <h2>MyBooks</h2>
    <ul class="books">
      <li *ngFor="let book of books " (click)="gotoDetail(book)">
              <span class="badge">{{book.id}}</span> {{book.name}}
      </li>
    </ul>
    <div>
      <router-outlet></router-outlet>
    </div>
       <div class="error" *ngIf="errorMessage">{{errorMessage}}</div>
```

7. Write the below code in **book.component.ts**:

```
import { Component, OnInit } from'@angular/core';
import { Router } from '@angular/router';
import { Book } from './book';
import { BookService } from './book.service';
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
```

```
books: Book[]=[];
errorMessage!: string;
constructor(private bookService: BookService, private router: Router) { }
getBooks() {
  this.bookService.getBooks().subscribe({
     next: books => {console.log(books);this.books = books},
     error:error => this.errorMessage = <any>error
   })
 }
 gotoDetail(book: Book): void {
  this.router.navigate(['/books/detail/', book.id]);
}
ngOnInit(): void{
  this.getBooks();
}
}
```

8. Update **book-detail.component.html** as below:

```
<div *ngIf="book">
<h2>{{ book.name }} details!</h2>
<div><label>id: </label>{{ book.id }}</div>
<div>
<label>name: </label> <input [(ngModel)]="book.name" placeholder="name" />
</div>
<button (click)="goBack()">Back</button>
</div>
```

9. Update **book-detail.component.ts** as below:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css'],
})
export class BookDetailComponent implements OnInit {
  book!: Book;
  error!: any;
  constructor(
    private bookService: BookService,
    private route: ActivatedRoute
  ) { }
  ngOnInit() {
      this.route.paramMap.subscribe(params => {
      this.bookService.getBook(params.get('id')).subscribe((book) => {
       this.book = book ?? this.book;
      });
    });
  }
  goBack() {
    window.history.back();
  }
}
```

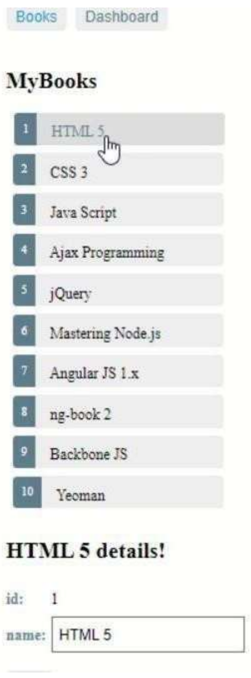
10. Update **dashboard.component.html** with below:

```
<h3>Top Books</h3>
<div class="grid grid-pad">
  <div *ngFor="let book of books" (click)="gotoDetail(book)" class="col-1-4">
    <div class="module book">
      <h4>{{ book.name }}</h4>
    </div>
  </div>
</div>
```

11. Update **dashboard.component.ts** with below:

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
 import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  books: Book[] = [];
  constructor(
    private router: Router,
    private bookService: BookService) { }
  ngOnInit(): void {
    this.bookService.getBooks()
      .subscribe(books => this.books = books.slice(1, 5));
  }
  gotoDetail(book: Book): void {
    this.router.navigate(['/books/detail', book.id]);
  }
}
```

12. Save the files and check the output in the browse



MyBooks

1 HTML 5
2 CSS 3
3 Java Script
4 Ajax Programming
5 jQuery
6 Mastering Node.js
7 Angular JS 1.x
8 ng-book 2
9 Backbone JS
10 Yeoman

HTML 5 details!

id: 1
name: HTML 5

## Excícisc-11(a)

**Course Name:** MongoDB Essentials - A Complete MongoDB Guide

**Module Name:**Installing MongoDB on the local computer, Create MongoDB Atlas Cluster

**Installing MongoDB on Local Computer:**

▪ **Download MongoDB:** Visit the official MongoDB website (https://www.mongodb.com/try/download/community) to download theMongoDB Community Server for your operating system.

▪ Install MongoDB:

• Windows: Run the downloaded installer and follow theinstallation wizard's instructions.

• macOS: Use Homebrew to install MongoDB by running **brewtap mongodb/brew** followed by **brew install mongodb/brew/mongodb-community**.

• Linux: Follow the instructions provided on the MongoDBwebsite for your specific distribution.

▪ Start MongoDB:

• Windows: MongoDB is typically installed as a service and startsautomatically. You can also manually start it using the Services application.

• macOS and Linux: Start MongoDB using the terminal by running

• mongod**.**

▪ **Access MongoDB Shell:**

• Open a new terminal window.
• Run **mongo** to start the MongoDB shell and interact with the database.

• Creating a MongoDB Atlas Cluster:

- MongoDB Atlas is a cloud-based MongoDB service that provides fully manageddatabases.

- ## Sign Up for MongoDB Atlas:

○ Go to the MongoDB Atlas website (https://www.mongodb.com/cloud/atlas) and click "Get started free."

○ Follow the prompts to sign up for an account.

### 1. Create a Cluster:

- After signing up and logging in, click the "Build a New Cluster"button.
- Choose a cloud provider, region, and cluster settings. You can

choosethe free tier (M0) to get started.

### 2. Configure Security:

- Click "Security" in the left-hand menu.
- Add your IP address to the IP Whitelist to allow connections to yourcluster.

### 3. Connect to Your Cluster:

- Click "Clusters" in the left-hand menu.
- Click the "Connect" button for your cluster.
- Choose "Connect Your Application" to get the connection string.

### 4. Connect to Atlas Cluster using MongoDB Shell:

- Open a terminal window.
- Run **mongo "your_connection_string"** to connect to your Atlascluster. Replace

**"your_connection_string"** with the actual connection string you obtained

## Exercise-11(b)

**Module Name:** Introduction to the CRUD Operations Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(),remove().

**SOURCE CODE:**

1. **Create (Insert) Documents**:

To insert documents into a collection, you can use the **insertOne()** or **insertMany()** methods.

```
// Insert a single document

db.collectionName.insertOne({ field1: value1, field2: value2 })

// Insert multiple documents
db.collectionName.insertMany([

{ field1: value1, field2: value2 },

{ field1: value3, field2: value4 }

])
```

2. Read (Find) Documents**:**

To retrieve documents froma collection, you can use the **find()** method.

```
// Find all documents in a collection
db.collectionName.find()

// Find documents with a specific field
valuedb.collectionName.find({ field: value })

// Find documents with a field value matching a rangedb.collectionName.find({ age: {
$gt: 25,

$lt: 40 } })
```

## 3. Update Documents:

To update documents in a collection, you can use the **updateOne()** or

**updateMany()**

methods.

```
db.collectionName.updateOne(
{ field: value }, { $set: { updatedField:
newValue } } )

// Update multiple documents
db.collectionName.updateMany(
{ field: value },
{ $set: { updatedField: newValue } }
)
```

## 4. Delete (Remove) Documents:

To remove documents froma collection, you can use the **deleteOne()** or

**deleteMany()** methods.

```
db.collectionName.deleteOne({ field: value })

db.collectionName.deleteMany({ field: value })
```

## Exercise-12(a)

**Module Name:**Create and Delete Databases and Collections

Write MongoDB queries to Create and drop databases and collections.

To create a new database, you can use the **use** command in the MongoDB shell.However, note that a database isn't actually created until you insert data into it.

```
// Switch to a new database (creates
it if it doesn't exist)
use newDatabaseName
```

### 1. Create a Collection:

Collections are created automatically when you insert data into them. However, youcan explicitly create a collection using the **createCollection()** method.

```
// Create a new collection in the current
databasedb.createCollection("newCollectionName")
```

### 2. Drop a Collection:

To drop (delete) a collection, you can use the **drop()** method.

```
// Drop a collection
db.collectionName.drop()
```

### 3. Drop a Database:

To drop a database, you can use the **dropDatabase()** method. Make sure to switchto the appropriate database before using this command.

```
// Drop the current database (make sure you're in the right database)

db.dropDatabase().
```

## Exercise-12(b)

**Module Name:** Introduction to MongoDB Queries

Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate()..

## SOURCE CODE:

**find()**: The **find()** method retrieves documents from a collection that match aspecified query. You can use various query operators to filter the results.

```
// Find all documents in a collection
db.collectionName.find()

// Find documents with a specific field
valuedb.collectionName.find({ field: value })

// Find documents with a field value matching a range
db.collectionName.find({ age: {
$gt: 25,

$lt: 40 } })\
```

**limit()**: The **limit()** method restricts the number of documents returned bya query.

```
// Limit the number of documents
returned
db.collectionName.find().limit(10)
```

**sort()**: The **sort()** method arranges the documents in a specific order based on oneor more fields.

```
// Sort documents in ascending order by a
fielddb.collectionName.find().sort({ field: 1
})
```

// Sort documents in descending order by a
fielddb.collectionName.find().sort({ field: - 1 })

**createIndex()**: The **createIndex()** method creates an index on specified fields in acollection, which can improve query performance.

// create an ascending index on a field
db.collectionName.
createIndex({
field: 1 })

// create a descending index on a field
db.collectionName.
createIndex({
field: - 1 })

**aggregate()**: The **aggregate()** method performs advanced data processing using apipeline of stages. It's useful for complex querying and transformations.

// Example aggregation pipeline: calculate average age by

genderdb.collectionName.aggregate([
{ $group: { _id: "$gender", avgAge: { $avg: "$age" } } }